Lab 10: Implementation of Binary Search Tree

Name and Enrolment :Syed Muhammad Raza Ali (02-134231-028)

Insert elements in a binary search tree
Traverse the binary search tree
Delete elements from binary search tree
Search elements in binary search tree

Exercise 1: Product Management

Zai Electronics is a retail store that has wide range of electronic items. The store wants to efficiently manage and organize the products based on their prices. Information about product such as Product id, Product name and price are required .Construct a program by applying Binary search tree to organize the products with following function.

a. Insert a product according to price in Binary search tree.{Insert 10 products}
b. Display all products by applying preorder, post order and in order traversal.
c. Search product having lowest price
d. Search product having highest price
e. Search product with price provided by customer.
f. Delete a product which is out of stock.

Code:

```
#include <iostream>
using namespace std;

struct bst_node {
	string productId;
	string productName;
	int productPrice;
	bst_node* left;
	bst_node* right;
};
bst_node* createNode(string productId,string productName,int producePrice) {
	bst_node* root = new bst_node();
	root->productId = productId;
	root->productName = productName;
	root->productPrice = producePrice;
	root->left = NULL;
	root->right = NULL;
	return root;
}
bst_node* insertProduct(bst_node* root, string productId, string productName, int
productPrice) {
```

```
	if (root == NULL) {
		bst_node* root = createNode(productId, productName, productPrice);
		return root;
	}
	else if (root->productPrice > productPrice) {
		root->left = insertProduct(root->left, productId, productName,
productPrice);
		return root;
	}
	else {
		root->right = insertProduct(root->right, productId, productName,
productPrice);
		return root;
	}
}

void preorderTraversal(bst_node* root) {
	if (root == NULL) {
		return;
	}
	else {
		cout << "ProductId : " << root->productId << endl
			<< "ProductName : " << root->productName << endl
			<< "ProductPrice : " << root->productPrice << endl;
		preorderTraversal(root->left);
		preorderTraversal(root->right);
	}
}

void inorderTraversal(bst_node* root) {
	if (root == NULL)
		return;
	else {
		inorderTraversal(root->left);
		cout << "ProductId : " << root->productId << endl
			<< "ProductName : " << root->productName << endl
			<< "ProductPrice : " << root->productPrice << endl;
		inorderTraversal(root->right);
	}
}

void postorderTraversal(bst_node* root) {
	if (root == NULL)
		return;
	else {
		postorderTraversal(root->left);
		postorderTraversal(root->right);
		cout << "ProductId : " << root->productId << endl
			<< "ProductName : " << root->productName << endl
			<< "ProductPrice : " << root->productPrice << endl;
	}
}

bst_node* findMinimumPrice(bst_node* root) {
	bst_node* temp = root;
	while (temp->left != NULL) {
		temp = temp->left;
	}
	return temp;
}
```

```
bst_node* findMaximumPrice(bst_node* root) {
	bst_node* temp = root;
	while (root->right != NULL)
		temp = temp->right;
	return root;
}
bst_node* searchProduct(bst_node* root, int productPrice) {
	if (root == NULL) {
		cout << "Product doesn't exist...\n";
		return root;
	}
	else if(root->productPrice == productPrice){
		cout << "Productfound...\nProductId : " << root->productId << endl
			<< "ProductName : " << root->productName << endl
			<< "ProductPrice : " << root->productPrice << endl;
	}
	else if (root->productPrice > productPrice) {
		searchProduct(root->left, productPrice);

	}
	else {
		searchProduct(root->right, productPrice);
	}
}
bst_node* deleteNode(bst_node* root, int key) {
	if (root == NULL)
		return root;

	if (key < root->productPrice)
		root->left = deleteNode(root->left, key);
	else if (key > root->productPrice)
		root->right = deleteNode(root->right, key);
	else {
		// Case 1: bst_node with only one child or no child
		if (root->left == NULL) {
			bst_node* temp = root->right;
			delete root;
			return temp;
		}
		else if (root->right == NULL) {
			bst_node* temp = root->left;
			delete root;
			return temp;
		}

		// Case 2: bst_node with two children
		bst_node* temp = findMinimumPrice(root->right);
		root->productPrice = temp->productPrice;
		root->right = deleteNode(root->right, temp->productPrice);
	}
	return root;
}
int main() {
	cout << "============== Zai Electronics ============== \n";
	bst_node*root = createNode("12199Z", "Speakers", 3400);
	insertProduct(root, "23729V", "Mouse", 1200);
	insertProduct(root, "11198A", "Keyboard", 4500);
	insertProduct(root, "23729X", "Monitor", 7599);
	insertProduct(root, "23729P", "usb", 800);
	cout << "============== Preorder ============== \n";
```

```
    preorderTraversal(root);
    cout << "============== Inorder ============== \n";
    inorderTraversal(root);
    cout << "============== Postorder ============== \n";
    postorderTraversal(root);
    cout << "============== Search Product ============== \n";
    searchProduct(root, 4500);
    cout << "============== Delete Product ============== \n";
    deleteNode(root, 1200);
    cout <<"--";
    preorderTraversal(root);
    return 0;
}
```

Output:

```
============== Preorder ==============
ProductId : 12199Z
ProductName : Speakers
ProductPrice : 3400
ProductId : 23729V
ProductName : Mouse
ProductPrice : 1200
ProductId : 23729P
ProductName : usb
ProductPrice : 800
ProductId : 11198A
ProductName : Keyboard
ProductPrice : 4500
ProductId : 23729X
ProductName : Monitor
ProductPrice : 7599
============== Inorder ==============
ProductId : 23729P
ProductName : usb
ProductPrice : 800
ProductId : 23729V
ProductName : Mouse
ProductPrice : 1200
ProductId : 12199Z
ProductName : Speakers
ProductPrice : 3400
ProductId : 11198A
ProductName : Keyboard
ProductPrice : 4500
ProductId : 23729X
ProductName : Monitor
ProductPrice : 7599
============== Postorder ==============
ProductId : 23729P
ProductName : usb
ProductPrice : 800
ProductId : 23729V
ProductName : Mouse
ProductPrice : 1200
ProductId : 23729X
ProductName : Monitor
ProductPrice : 7599
ProductId : 11198A
ProductName : Keyboard
ProductPrice : 4500
ProductId : 12199Z
ProductName : Speakers
ProductPrice : 3400
============== Search Product
```

```
============== Search Product ==============
Productfound...
ProductId : 11198A
ProductName : Keyboard
ProductPrice : 4500
============== Delete Product ==============
--ProductId : 23729X
ProductName : Monitor
ProductPrice : 7599
ProductId : 11198A
ProductName : Keyboard
ProductPrice : 4500
ProductId : 12199Z
ProductName : Speakers
ProductPrice : 3400
ProductId : 23729V
ProductName : Mouse
ProductPrice : 1200
ProductId : 23729P
ProductName : usb
ProductPrice : 800
```

Exercise 2: Computer Logins

Consider the problem of organizing a collection of computer user-ids and passwords. Each time a user logs in to the system by entering his or her user-id and a secret password, the system must check the validity of this user-id and password to verify that this is a legitimate user. Because this user validation must be done many times each day, it is necessary to structure this information in such a way that it can be searched rapidly.

Write a program that implements the above scenario using binary search tree with following functions

a. Insert at least 10 user with password.
b. Display all users with their passwords.

c. Search for legitimate user. Display a message for found and not found.

d. Delete the user.

Code:

```
#include <iostream>
using namespace std;

struct bst_node {
	string userId;
	string password;
	int userNumber;
	bst_node* left;
	bst_node* right;
};
bst_node* createNode(string userId, string password,int userNumber) {
	bst_node* root = new bst_node();
	root->userId = userId;
	root->password = password;
	root->userNumber = userNumber;
	root->left = NULL;
	root->right = NULL;
	return root;
}
bst_node* insertUser(bst_node* root, string userId, string password, int userNumber) {
	if (root == NULL) {
		bst_node* root = createNode(userId, password, userNumber);
		return root;
	}
	else if (root->userNumber > userNumber) {
		root->left = insertUser(root->left, userId, password, userNumber);
		return root;
	}
	else {
		root->right = insertUser(root->right, userId, password, userNumber);
		return root;
	}
}

void preorderTraversal(bst_node* root) {
	if (root == NULL) {
		return;
	}
	else {
		cout << "userId : " << root->userId << endl
			<< "password : " << root->password << endl
			<< "userNumber : " << root->userNumber << endl;
		preorderTraversal(root->left);
		preorderTraversal(root->right);
	}
}

void inorderTraversal(bst_node* root) {
	if (root == NULL)
		return;
	else {
		inorderTraversal(root->left);
		cout << "userId : " << root->userId << endl
```

```
                        << "password : " << root->password << endl
                        << "userNumber : " << root->userNumber << endl;
                inorderTraversal(root->right);
        }
}

void postorderTraversal(bst_node* root) {
        if (root == NULL)
                return;
        else {
                postorderTraversal(root->left);
                postorderTraversal(root->right);
                cout << "userId : " << root->userId << endl
                        << "password : " << root->password << endl
                        << "userNumber : " << root->userNumber << endl;
        }
}

bst_node* findMinimumPrice(bst_node* root) {
        bst_node* temp = root;
        while (temp->left != NULL) {
                temp = temp->left;
        }
        return temp;
}
bst_node* findMaximumPrice(bst_node* root) {
        bst_node* temp = root;
        while (root->right != NULL)
                temp = temp->right;
        return root;
}
bst_node* searchUser(bst_node* root, int userNumber) {
        if (root == NULL) {
                cout << "user doesn't exist...\n";
                return root;
        }
        else if (root->userNumber == userNumber) {
                cout << "Userfound...\nuserId : " << root->userId << endl
                        << "userPassword : " << root->password << endl
                        << "userNumber : " << root->userNumber << endl;
        }
        else if (root->userNumber > userNumber) {
                searchUser(root->left, userNumber);

        }
        else {
                searchUser(root->right, userNumber);
        }
}
bst_node* deleteNode(bst_node* root, int key) {
        if (root == NULL)
                return root;

        if (key < root->userNumber)
                root->left = deleteNode(root->left, key);
        else if (key > root->userNumber)
                root->right = deleteNode(root->right, key);
        else {
                // Case 1: bst_node with only one child or no child
                if (root->left == NULL) {
```

```
                bst_node* temp = root->right;
                delete root;
                return temp;
            }
            else if (root->right == NULL) {
                bst_node* temp = root->left;
                delete root;
                return temp;
            }

            // Case 2: bst_node with two children
            bst_node* temp = findMinimumPrice(root->right);
            root->userNumber = temp->userNumber;
            root->right = deleteNode(root->right, temp->userNumber);
        }
        return root;
}

int main() {
        cout << "==================== Insert Users ====================\n";
        bst_node*root = createNode("21128", "dbuaksefi99", 10);
        insertUser(root, "32872", "sjdhsd83e8", 64);
        insertUser(root, "23998", "3ennsyd8y", 22);
        insertUser(root, "33983", "rdjs3343", 83);
        insertUser(root, "30293", "fhdru383",3);
        insertUser(root, "923874", "cnsori39843", 9);
        cout << "==================== Preorder Traversal ====================\n";
        preorderTraversal(root);
        cout << "==================== Search Users ====================\n";
        searchUser(root, 22);
        searchUser(root, 3);
        searchUser(root, 2112);
        cout << "==================== Delete Users ====================\n";
        deleteNode(root, 83);
        preorderTraversal(root);
        return 0;
 }
```

Output:

```
==================== Preorder Traversal ====================
userId : 21128
password : dbuaksefi99
userNumber : 10
userId : 30293
password : fhdru383
userNumber : 3
userId : 923874
password : cnsori39843
userNumber : 9
userId : 32872
password : sjdhsd83e8
userNumber : 64
userId : 23998
password : 3ennsyd8y
userNumber : 22
userId : 33983
password : rdjs3343
userNumber : 83
==================== Search Users ====================
Userfound...
userId : 23998
userPassword : 3ennsyd8y
userNumber : 22
Userfound...
userId : 30293
userPassword : fhdru383
userNumber : 3
user doesn't exist...
==================== Delete Users ====================
userId : 21128
password : dbuaksefi99
userNumber : 10
userId : 30293
password : fhdru383
userNumber : 3
userId : 923874
password : cnsori39843
userNumber : 9
userId : 32872
password : sjdhsd83e8
userNumber : 64
userId : 23998
password : 3ennsyd8y
userNumber : 22
```

Exercise 3: Student Report

Write a program that inserts the following marks of 15 students in a binary search tree in a way those who scores above average will be organized in right side and those who scores less than average will be stored in left side of the binary search tree. Average score of the students is 13.6

10.5	14.5	9.0	12.0	14.0	16.0	8.0	10.0	11.5	13.0	15.0	18.0	17.5	19.0	17.0

a. Insert the marks.
b. Display marks in sorted order
c. Display number of students that scores above average.
d. Display number of students that scores below average.
e. Display the highest marks.
f. Display the lowest marks.
g. Search the marks provided by user.
h. Delete the marks provided by user.

Code:

```
#include <iostream>
using namespace std;

struct bst_node {
	float marks;
	bst_node* left;
	bst_node* right;
};
bst_node* createNode(float marks) {
	bst_node* root = new bst_node();
	root->marks = marks;
	root->left = NULL;
	root->right = NULL;
	return root;
}
bst_node* insertMarks(bst_node* root,float marks) {
	if (root == NULL) {
		bst_node* root = createNode( marks);
		return root;
	}
	else if (root->marks > marks) {
		root->left = insertMarks(root->left, marks);
		return root;
	}
	else {
		root->right = insertMarks(root->right, marks);
		return root;
	}
}

void preorderTraversal(bst_node* root) {
	if (root == NULL) {
		return;
	}
```

```
	else {
			cout << "marks : " << root->marks << endl;
		preorderTraversal(root->left);
		preorderTraversal(root->right);
	}
}

void inorderTraversal(bst_node* root) {
	if (root == NULL)
		return;
	else {
		inorderTraversal(root->left);
			cout << "marks : " << root->marks << endl;
		inorderTraversal(root->right);
	}
}

void postorderTraversal(bst_node* root) {
	if (root == NULL)
		return;
	else {
		postorderTraversal(root->left);
		postorderTraversal(root->right);
			cout << "marks : " << root->marks << endl;
	}
}
void greaterThanAverage(bst_node* root) {
	if (root == NULL)
		return;
	else {
		if (root->marks > 13.6)
			cout << root->marks << " ";

		greaterThanAverage(root->right);
	}
}
void lesserThanAverage(bst_node* root) {
	if (root == NULL)
		return;
	else {
		if(root->marks<13.6)
		cout << root->marks << " ";
		lesserThanAverage(root->left);
	}
}
bst_node* findMinimumMarks(bst_node* root) {
	bst_node* temp = root;
	while (temp->left != NULL) {
		temp = temp->left;
	}
	return temp;
}
bst_node* findMaximumMarks(bst_node* root) {
	bst_node* temp = root;
	while (temp->right != NULL)
		temp = temp->right;
	return temp;
}
bst_node* searchProduct(bst_node* root, float marks) {
	if (root == NULL) {
```

```
			cout << "Student doesn't exist...\n";
			return root;
		}
		else if (root->marks == marks) {
			cout << "Studentfound...\n"
				<< "marks : " << root->marks << endl;
		}
		else if (root->marks < marks) {
			searchProduct(root->left, marks);

		}
		else {
			searchProduct(root->right, marks);
		}
}
bst_node* deleteNode(bst_node* root, float key) {
		if (root == NULL)
			return root;

		if (key < root->marks)
			root->left = deleteNode(root->left, key);
		else if (key > root->marks)
			root->right = deleteNode(root->right, key);
		else {
			// Case 1: bst_node with only one child or no child
			if (root->left == NULL) {
				bst_node* temp = root->right;
				delete root;
				return temp;
			}
			else if (root->right == NULL) {
				bst_node* temp = root->left;
				delete root;
				return temp;
			}

			// Case 2: bst_node with two children
			bst_node* temp = findMinimumMarks(root->right);
			root->marks = temp->marks;
			root->right = deleteNode(root->right, temp->marks);
		}
		return root;
}

int main() {
		bst_node* root = createNode(13.6);
		insertMarks(root, 10.5);
		insertMarks(root, 14.5);
		insertMarks(root, 9.0);
		insertMarks(root, 12.0);
		insertMarks(root, 14.0);
		insertMarks(root, 16.0);
		insertMarks(root, 8.0);
		insertMarks(root, 10.0);
		insertMarks(root, 11.5);
		insertMarks(root, 13.0);
		insertMarks(root, 15.0);
		insertMarks(root, 18.0);
		insertMarks(root, 17.5);
		insertMarks(root, 19.0);
```

```
    insertMarks(root, 17.0);
    cout << "============== Marks in Sorted order ============== \n";
    inorderTraversal(root);
    cout << "============== Highest Marks ============== \n";
    cout <<findMaximumMarks(root)->marks;
    cout << "\n============== Lowest Marks ============== \n";
    cout << findMinimumMarks(root)->marks;
    cout << "\n============== Greater than average ============== \n";
    greaterThanAverage(root);
    cout << "\n============== Lesser than average ============== \n";
    lesserThanAverage(root);
    cout << "\n============== Delete Node ============== \n";
    deleteNode(root, 16.0);
    inorderTraversal(root);
    return 0;
}
```

Output:

```
============== Marks in Sorted order ==============
marks : 8
marks : 9
marks : 10
marks : 10.5
marks : 11.5
marks : 12
marks : 13
marks : 13.6
marks : 14
marks : 14.5
marks : 15
marks : 16
marks : 17
marks : 17.5
marks : 18
marks : 19
============== Highest Marks ==============
19
============== Lowest Marks ==============
8
============== Greater than average ==============
13.6 14.5 16 18 19
============== Lesser than average ==============
10.5 9 8
============== Delete Node ==============
marks : 8
marks : 9
marks : 10
marks : 10.5
marks : 11.5
marks : 12
marks : 13
marks : 13.6
marks : 14
marks : 14.5
marks : 15
marks : 17
marks : 17.5
marks : 18
marks : 19
```