**Lab Manual 04**

**DATA STRUCTURE**

Submitted by: **Tahira Inam**

Roll No: 105

Instructor: Azka Mir

3rd Semester (Blue)

Department of Software Engineering

University of Sialkot

Contents

[Objectives 3](#_Toc187586916)

[Exercise 3.1: Family Tree Structure. 3](#_Toc187586917)

[Source Code and Outputs 3](#_Toc187586918)

[Exercise 4 .1 Code 3](#_Toc187586919)

[Output: 10](#_Toc187586920)

## Objectives

1. Understanding the concepts of Binary Tree and its operations.
2. Implement CRUD operations.

### Exercise 3.1: Family Tree Structure.

Implement a family tree in c++ using a tree structure, where each node represents a family member, and the relations are determined by yes/no questions about family side (e.g mother's side vs. father's side). You need to implement the following CRUD operations and the search operation for the binary tree:

1. **Create:** add a new family member to the tree (either as a left or right child, representing either the mother's or father's side).

2. **Read:** display the entire family tree starting from the root, showing the family relations in a yes/no format (e.g is this person from your mother's side?).

3. **Update:** modify the details of an existing family member (e.g changing thier name or relation).

4. **Delete:** remove a family member from the tree(i.e delete a node from the tree).

5. **search:** search for a family member by name. if found, display their relation and position in the family tree.

6. **Additional requirement:** after implementing the CRUD operations and seach , allow the user to interact with the tree by answering yes/no questions to trace thier family tree, ending at a leaf node where the final family member is displayed.

## Source Code and Outputs

### Exercise 4 .1 Code

#include <iostream>

#include <string>

using namespace std;

struct FamilyMember {

string name;

string relation;

FamilyMember \*left;

FamilyMember \*right;

};

class FamilyTree {

private:

FamilyMember \*root;

public:

FamilyTree() {

root = nullptr;

}

void display(FamilyMember \*node) {

if (node) {

cout << node->name << " (" << node->relation << ")\n";

display(node->left);

display(node->right);

}

}

FamilyMember \*find(FamilyMember \*node, string name) {

if (!node || node->name == name)

return node;

FamilyMember \*found = find(node->left, name);

return found ? found : find(node->right, name);

}

FamilyMember \*remove(FamilyMember \*node, string name) {

if (!node)

return nullptr;

if (node->name == name) {

if (!node->left && !node->right) {

delete node;

return nullptr;

}

if (node->left && !node->right) {

FamilyMember \*temp = node->left;

delete node;

return temp;

}

if (!node->left && node->right) {

FamilyMember \*temp = node->right;

delete node;

return temp;

}

}

node->left = remove(node->left, name);

node->right = remove(node->right, name);

return node;

}

void add(string name, string relation, bool isMotherSide = true) {

FamilyMember \*newMember = new FamilyMember{name, relation, nullptr, nullptr};

if (!root) {

root = newMember;

cout << "Root member added: " << name << endl;

return;

}

FamilyMember \*current = root;

while (true) {

if (isMotherSide) {

if (!current->left) {

current->left = newMember;

cout << name << " added on mother's side." << endl;

break;

}

current = current->left;

} else {

if (!current->right) {

current->right = newMember;

cout << name << " added on father's side." << endl;

break;

}

current = current->right;

}

}

}

void showTree() {

if (!root) {

cout << "Family tree is empty." << endl;

return;

}

display(root);

}

void modify(string name, string newName, string newRelation) {

FamilyMember \*member = find(root, name);

if (member) {

member->name = newName;

member->relation = newRelation;

cout << "Member updated: " << newName << " (" << newRelation << ")" << endl;

} else {

cout << "Member not found!" << endl;

}

}

void removeMember(string name) {

root = remove(root, name);

}

void search(string name) {

FamilyMember \*member = find(root, name);

if (member) {

cout << "Found: " << member->name << " (" << member->relation << ")" << endl;

} else {

cout << "Member not found!" << endl;

}

}

};

int main() {

FamilyTree family;

int choice;

while (true) {

cout << "\n1. Add Member\n2. Show Tree\n3. Modify Member\n4. Remove Member\n5. Search Member\n6. Exit\nChoose an option: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1: {

string name, relation;

int isMotherSide;

cout << "Enter name: ";

getline(cin, name);

cout << "Enter relation: ";

getline(cin, relation);

cout << "Is it on mother's side? (1 for yes, 0 for no): ";

cin >> isMotherSide;

family.add(name, relation, isMotherSide);

break;

}

case 2:

family.showTree();

break;

case 3: {

string name, newName, newRelation;

cout << "Enter name to modify: ";

getline(cin, name);

cout << "Enter new name: ";

getline(cin, newName);

cout << "Enter new relation: ";

getline(cin, newRelation);

family.modify(name, newName, newRelation);

break;

}

case 4: {

string name;

cout << "Enter name to remove: ";

getline(cin, name);

family.removeMember(name);

break;

}

case 5: {

string name;

cout << "Enter name to search: ";

getline(cin, name);

family.search(name);

break;

}

case 6:

cout << "Exiting..." << endl;

return 0;

default:

cout << "Invalid choice! Try again." << endl;

}

}

return 0;

}

### Output:

