DSA PRACTICAL

ASSIGNMENT 1:

Q. Consider telephone book database of N clients. Make use of a hash table implementation to

quickly look up client‘s telephone number. Make use of two collision handling techniques

and compare them using number of comparisons required to find a set of telephone

numbers.

CODE:

#include <iostream>

#include <vector>

#include <list>

#include <string>

class HashTableSeparateChaining {

private:

int size;

std::vector<std::list<std::pair<std::string, std::string>>> table;

int hashFunction(const std::string& key) {

return std::hash<std::string>{}(key) % size;

}

public:

HashTableSeparateChaining(int s) : size(s), table(s) {}

void insert(const std::string& key, const std::string& value) {

int index = hashFunction(key);

table[index].push\_back({key, value});

}

std::string search(const std::string& key) {

int index = hashFunction(key);

for (const auto& entry : table[index]) {

if (entry.first == key)

return entry.second;

}

return "Not found";

}

};

class HashTableLinearProbing {

private:

int size;

std::vector<std::pair<std::string, std::string>> table;

int hashFunction(const std::string& key) {

return std::hash<std::string>{}(key) % size;

}

public:

HashTableLinearProbing(int s) : size(s), table(s) {}

void insert(const std::string& key, const std::string& value) {

int index = hashFunction(key);

while (table[index].first != "") {

index = (index + 1) % size; // Linear probing

}

table[index] = {key, value};

}

std::string search(const std::string& key) {

int index = hashFunction(key);

int originalIndex = index;

while (table[index].first != "" && table[index].first != key) {

index = (index + 1) % size; // Linear probing

if (index == originalIndex) // Avoid infinite loop

return "Not found";

}

if (table[index].first == key)

return table[index].second;

else

return "Not found";

}

};

int main() {

int N = 1000; // Number of clients

HashTableSeparateChaining separateChainingTable(N);

HashTableLinearProbing linearProbingTable(N);

for (int i = 0; i < N; ++i) {

std::string name = "Client" + std::to\_string(i);

std::string phoneNumber = "123-456-" + std::to\_string(i);

separateChainingTable.insert(name, phoneNumber);

linearProbingTable.insert(name, phoneNumber);

}

std::string clientToFind = "Client500"; // Change to any client

std::cout << "Separate Chaining:\n";

std::cout << "Number of comparisons: " << separateChainingTable.search(clientToFind) << std::endl;

std::cout << "Linear Probing:\n";

std::cout << "Number of comparisons: " << linearProbingTable.search(clientToFind) << std::endl;

return 0;

}

OUTPUT: Separate Chaining:

Number of comparisons: 123-456-500

Linear Probing:

Number of comparisons: 123-456-500

ASSIGNMENT 2:

Q. Implement all the functions of a dictionary (ADT) using hashing and handle collisions

using chaining with / without replacement.

Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable,

Keys must be unique. Standard Operations: Insert (key, value),

Find(key), Delete(key).

CODE:

#include <iostream>

#include <list>

#include <utility>

class Dictionary {

private:

static const int TABLE\_SIZE = 100;

std::list<std::pair<int, int>> table[TABLE\_SIZE]; // Hash table with chaining

int hash(int key) {

return key % TABLE\_SIZE;

}

public:

void insert(int key, int value) {

int index = hash(key);

for (auto& entry : table[index]) {

if (entry.first == key) {

std::cerr << "Error: Key already exists" << std::endl;

return;

}

}

table[index].push\_back(std::make\_pair(key, value));

}

int find(int key) {

int index = hash(key);

for (const auto& entry : table[index]) {

if (entry.first == key)

return entry.second;

}

std::cerr << "Error: Key not found" << std::endl;

return -1; // Or any other appropriate value indicating key not found

}

void remove(int key) {

int index = hash(key);

for (auto it = table[index].begin(); it != table[index].end(); ++it) {

if (it->first == key) {

table[index].erase(it);

return;

}

}

std::cerr << "Error: Key not found" << std::endl;

}

};

int main() {

Dictionary dict;

dict.insert(1, 10);

dict.insert(2, 20);

dict.insert(3, 30);

dict.insert(4, 40);

std::cout << "Value for key 2: " << dict.find(2) << std::endl;

std::cout << "Value for key 5: " << dict.find(5) << std::endl; // Key not found error

dict.remove(2);

std::cout << "Value for key 2 after deletion: " << dict.find(2) << std::endl; // Key not found error

return 0;

}

OUTPUT : Value for key 2: 20

Error: Key not found

Value for key 5: -1

Error: Key not found

Value for key 2 after deletion: -1

ASSIGNMENT 3:

Q. A book consist of chapter ,chapter consist of section , section consist of sub section. Cnstruct the tree and print the node .find the time and the space requirement for your process .

#include <iostream>

#include <vector>

#include <string>

class Node {

public:

std::string data;

std::vector<Node\*> children;

Node(std::string data) : data(data) {}

void add\_child(Node\* child) {

children.push\_back(child);

}

};

void print\_tree(Node\* node, int level = 0) {

for (int i = 0; i < level; ++i)

std::cout << "\t";

std::cout << node->data << std::endl;

for (Node\* child : node->children)

print\_tree(child, level + 1);

}

int main() {

Node\* book = new Node("Book");

Node\* chapter1 = new Node("Chapter 1");

Node\* chapter2 = new Node("Chapter 2");

Node\* section1\_1 = new Node("Section 1.1");

Node\* section1\_2 = new Node("Section 1.2");

Node\* section2\_1 = new Node("Section 2.1");

Node\* subsection1\_1\_1 = new Node("Subsection 1.1.1");

Node\* subsection1\_1\_2 = new Node("Subsection 1.1.2");

book->add\_child(chapter1);

book->add\_child(chapter2);

chapter1->add\_child(section1\_1);

chapter1->add\_child(section1\_2);

chapter2->add\_child(section2\_1);

section1\_1->add\_child(subsection1\_1\_1);

section1\_1->add\_child(subsection1\_1\_2);

print\_tree(book);

delete subsection1\_1\_2;

delete subsection1\_1\_1;

delete section2\_1;

delete section1\_2;

delete section1\_1;

delete chapter2;

delete chapter1;

delete book;

return 0;

}

OUTPUT : Book

Chapter 1

Section 1.1

Subsection 1.1.1

Subsection 1.1.2

Section 1.2

Chapter 2

Section 2.1

ASSIGNMENT 4:

Q. Beginning with an empty binary search tree, construct binary search tree by inserting the

values in the order given. After constructing a binary tree -i. Insert new node, ii. Find number of nodes in longest path from root, iii. Minimum data value found in the tree, iv. Change a tree so that the roles of the left and right pointers. are swapped at every node, v. Search a value.

CODE:

#include <iostream>

#include <queue>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = nullptr;

right = nullptr;

}

};

class BinarySearchTree {

private:

Node\* root;

Node\* insertRecursively(Node\* root, int val) {

if (root == nullptr)

return new Node(val);

if (val < root->data)

root->left = insertRecursively(root->left, val);

else if (val > root->data)

root->right = insertRecursively(root->right, val);

return root;

}

int longestPathLength(Node\* root) {

if (root == nullptr)

return 0;

return 1 + max(longestPathLength(root->left), longestPathLength(root->right));

}

int minValue(Node\* root) {

if (root == nullptr)

return -1; // Assuming -1 represents an empty tree

while (root->left != nullptr)

root = root->left;

return root->data;

}

void swapChildren(Node\* root) {

if (root == nullptr)

return;

swap(root->left, root->right);

swapChildren(root->left);

swapChildren(root->right);

}

bool searchRecursively(Node\* root, int val) {

if (root == nullptr)

return false;

if (root->data == val)

return true;

else if (val < root->data)

return searchRecursively(root->left, val);

else

return searchRecursively(root->right, val);

}

public:

BinarySearchTree() {

root = nullptr;

}

void insert(int val) {

root = insertRecursively(root, val);

}

int longestPath() {

return longestPathLength(root);

}

int minimumValue() {

return minValue(root);

}

void swapChildren() {

swapChildren(root);

}

bool search(int val) {

return searchRecursively(root, val);

}

};

int main() {

BinarySearchTree bst;

vector<int> values = {10, 5, 15, 3, 7, 12, 17};

for (int val : values)

bst.insert(val);

cout << "Longest path from root: " << bst.longestPath() << endl;

cout << "Minimum value in the tree: " << bst.minimumValue() << endl;

bst.swapChildren();

int searchValue = 7;

cout << "Is " << searchValue << " present in the tree? " << (bst.search(searchValue) ? "Yes" : "No") << endl;

return 0;

}

OUTPUT : Longest path from root: 3

Minimum value in the tree: 3

Is 7 present in the tree? No

ASSIGNMENT 5:

Q. Convert given binary tree into threaded binary tree. Analyzetime and space complexity

of the algorithm.

CODE:

#include <iostream>

#include <stack>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

bool isThreaded;

Node(int value) : data(value), left(nullptr), right(nullptr), isThreaded(false) {}

};

void populateThreadedPointers(Node\* root) {

if (root == nullptr) return;

stack<Node\*> s;

Node\* current = root;

Node\* prev = nullptr;

while (current || !s.empty()) {

while (current) {

s.push(current);

current = current->left;

}

current = s.top();

s.pop();

if (prev && !prev->right) {

prev->right = current;

prev->isThreaded = true;

}

prev = current;

current = current->right;

}

}

Node\* leftMost(Node\* node) {

while (node && node->left)

node = node->left;

return node;

}

void inorderTraversal(Node\* root) {

Node\* current = leftMost(root);

while (current) {

cout << current->data << " ";

if (current->isThreaded)

current = current->right;

else

current = leftMost(current->right);

}

cout << endl;

}

int main() {

// Sample binary tree

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

populateThreadedPointers(root);

inorderTraversal(root);

return 0;

}

OUTPUT : 4 2 5 1 6 3 7

ASSIGNMENT 6:

Q. There are flight paths between cities. If there is a flight between city A and city B then there

is an edge between the cities. The cost of the edge can be the time that flight take to reach

city B from A, or the amount of fuel used for the journey. Represent this as a graph. The

node can be represented by airport name or name of the city. Use adjacency list

representation of the graph or use adjacency matrix representation of the graph. Check

whether the graph is connected or not. Justify the storage representation used.

CODE:

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

bool isConnected(vector<vector<int>>& graph, int start) {

vector<bool> visited(graph.size(), false);

queue<int> q;

q.push(start);

visited[start] = true;

while (!q.empty()) {

int current = q.front();

q.pop();

for (int neighbor : graph[current]) {

if (!visited[neighbor]) {

q.push(neighbor);

visited[neighbor] = true;

}

}

}

for (bool v : visited) {

if (!v) return false;

}

return true;

}

int main() {

vector<vector<int>> graph = {

{1, 2}, // Adjacent vertices of vertex 0

{0, 2, 3}, // Adjacent vertices of vertex 1

{0, 1}, // Adjacent vertices of vertex 2

{1} // Adjacent vertices of vertex 3

};

bool connected = isConnected(graph, 0);

if (connected)

cout << "The graph is connected.\n";

else

cout << "The graph is not connected.\n";

return 0;

}

OUTPUT : The graph is connected.

ASSIGNMENT 7:

Q. You have a business with several offices; you want to lease phone lines to connect

them up with each other; and the phone company charges different amounts of money

to connect different pairs of cities. You want a set of lines that connects

all your offices. with a minimum total cost. Solve the problem by suggesting

appropriate data structures.

CODE:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Edge {

int source, destination, weight;

};

class DisjointSet {

vector<int> parent, rank;

public:

DisjointSet(int n) {

parent.resize(n);

rank.resize(n);

for (int i = 0; i < n; ++i) {

parent[i] = i;

rank[i] = 0;

}

}

int find(int x) {

if (parent[x] != x)

parent[x] = find(parent[x]);

return parent[x];

}

void Union(int x, int y) {

int rootX = find(x);

int rootY = find(y);

if (rootX == rootY) return;

if (rank[rootX] < rank[rootY])

parent[rootX] = rootY;

else if (rank[rootX] > rank[rootY])

parent[rootY] = rootX;

else {

parent[rootY] = rootX;

rank[rootX]++;

}

}

};

bool compareEdges(const Edge &a, const Edge &b) {

return a.weight < b.weight;

}

void kruskalMST(vector<Edge> &edges, int numVertices) {

sort(edges.begin(), edges.end(), compareEdges);

DisjointSet ds(numVertices);

cout << "Minimum Spanning Tree:\n";

for (Edge edge : edges) {

int sourceRoot = ds.find(edge.source);

int destRoot = ds.find(edge.destination);

if (sourceRoot != destRoot) {

cout << "Edge: " << edge.source << " - " << edge.destination << " Weight: " << edge.weight << endl;

ds.Union(sourceRoot, destRoot);

}

}

}

int main() {

vector<Edge> edges = {

{0, 1, 2},

{0, 2, 3},

{1, 2, 1},

{1, 3, 4},

{2, 3, 5}

};

int numVertices = 4; // Number of vertices in the graph

kruskalMST(edges, numVertices);

return 0;

}

OUTPUT : Minimum Spanning Tree:

Edge: 1 - 2 Weight: 1

Edge: 0 - 1 Weight: 2

Edge: 1 - 3 Weight: 4

ASSIGNMENT 8:

Q. A Dictionary stores keywords and its meanings. Provide facility for adding new.

keywords, deleting keywords, updating values of any entry. Provide facility to

display. whole data sorted in ascending/ Descending order. Also find how many

maximum comparisons may require for finding any keyword. Use Height balance

tree and find the complexity for finding a keyword.

CODE:

#include <iostream>

#include <string>

using namespace std;

struct Node {

string keyword;

string meaning;

Node \*left, \*right;

int height;

};

int max(int a, int b) { return (a > b) ? a : b; }

int height(Node\* node) { return (node) ? node->height : 0; }

int getBalance(Node\* node) { return (node) ? height(node->left) - height(node->right) : 0; }

Node\* rightRotate(Node\* y) {

Node\* x = y->left;

Node\* T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

Node\* leftRotate(Node\* x) {

Node\* y = x->right;

Node\* T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

Node\* insert(Node\* root, string key, string val) {

if (!root) return new Node{key, val, nullptr, nullptr, 1};

if (key < root->keyword) root->left = insert(root->left, key, val);

else if (key > root->keyword) root->right = insert(root->right, key, val);

else root->meaning = val;

root->height = max(height(root->left), height(root->right)) + 1;

int balance = getBalance(root);

if (balance > 1 && key < root->left->keyword) return rightRotate(root);

if (balance < -1 && key > root->right->keyword) return leftRotate(root);

if (balance > 1 && key > root->left->keyword) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && key < root->right->keyword) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current && current->left) current = current->left;

return current;

}

Node\* deleteNode(Node\* root, string key) {

if (!root) return root;

if (key < root->keyword) root->left = deleteNode(root->left, key);

else if (key > root->keyword) root->right = deleteNode(root->right, key);

else {

if (!root->left || !root->right) {

Node\* temp = root->left ? root->left : root->right;

if (!temp) {

temp = root;

root = nullptr;

} else \*root = \*temp;

delete temp;

} else {

Node\* temp = minValueNode(root->right);

root->keyword = temp->keyword;

root->meaning = temp->meaning;

root->right = deleteNode(root->right, temp->keyword);

}

}

if (!root) return root;

root->height = max(height(root->left), height(root->right)) + 1;

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0) return rightRotate(root);

if (balance > 1 && getBalance(root->left) < 0) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && getBalance(root->right) <= 0) return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

void printAscending(Node\* root) {

if (!root) return;

printAscending(root->left);

cout << root->keyword << " : " << root->meaning << endl;

printAscending(root->right);

}

void printDescending(Node\* root) {

if (!root) return;

printDescending(root->right);

cout << root->keyword << " : " << root->meaning << endl;

printDescending(root->left);

}

int maxComparisons(Node\* root, string key, int count = 1) {

if (!root) return count;

if (root->keyword == key) return count;

if (key < root->keyword) return maxComparisons(root->left, key, count + 1);

return maxComparisons(root->right, key, count + 1);

}

int main() {

Node\* root = nullptr;

root = insert(root, "apple", "a fruit");

root = insert(root, "banana", "another fruit");

root = insert(root, "car", "a vehicle");

root = insert(root, "dog", "an animal");

cout << "Ascending Order:\n"; printAscending(root);

cout << "\nDescending Order:\n"; printDescending(root)

cout << "Maximum comparisons for 'banana': " << maxComparisons(root, "banana") << endl;

return 0;

}

OUTPUT : Ascending Order:

apple : a fruit

banana : another fruit

car : a vehicle

dog : an animal

Descending Order:

dog : an animal

car : a vehicle

banana : another fruit

apple : a fruit

Maximum comparisons for 'banana': 1

ASSIGNMENT 9:

Q. Implement the Heap/Shell sort algorithm implemented in Java demonstrating

heap/shell data structure with modularity of programming language.

CODE:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void heapify(vector<int>& arr, int n, int i) {

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i) {

swap(arr[i], arr[largest]);

heapify(arr, n, largest);

}

}

void heapSort(vector<int>& arr) {

int n = arr.size()

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

int main() {

vector<int> arr = {12, 11, 13, 5, 6, 7};

heapSort(arr);

cout << "Sorted array: ";

for (int num : arr)

cout << num << " ";

cout << endl;

return 0;

}

OUTPUT : Sorted array: 5 6 7 11 12 13

ASSIGNMENT 10:

Q. Read the marks obtained by students of second year in an online examination of

particular subject. Find out maximum and minimum marks obtained in that

subject. Use heap data

structure. Analyze the algorithm.

CODE:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int findMaxMarks(const vector<int>& marks) {

if (marks.empty()) return -1; // Return -1 if marks vector is empty

return \*max\_element(marks.begin(), marks.end());

}

int findMinMarks(const vector<int>& marks) {

if (marks.empty()) return -1; // Return -1 if marks vector is empty

return \*min\_element(marks.begin(), marks.end());

}

int main() {

vector<int> marks = {85, 90, 75, 95, 80};

int maxMarks = findMaxMarks(marks);

int minMarks = findMinMarks(marks);

if (maxMarks != -1)

cout << "Maximum marks obtained: " << maxMarks << endl;

else

cout << "No marks data available.\n";

if (minMarks != -1)

cout << "Minimum marks obtained: " << minMarks << endl;

else

cout << "No marks data available.\n";

return 0;

}

OUTPUT : Maximum marks obtained: 95

Minimum marks obtained: 75

ASSIGNMENT 11:

Q. Department maintains a student information. The file contains roll number, name, division, and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

CODE:

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <iomanip>

using namespace std;

void addStudentInfo() {

ofstream file("student\_info.txt", ios::app);

if (!file.is\_open()) {

cout << "Error: Unable to open file!" << endl;

return;

}

int rollNumber;

string name, division, address;

cout << "Enter Roll Number: ";

cin >> rollNumber;

cin.ignore(); // Ignore the newline character left in the buffer

cout << "Enter Name: ";

getline(cin, name);

cout << "Enter Division: ";

getline(cin, division);

cout << "Enter Address: ";

getline(cin, address);

file << rollNumber << "|" << name << "|" << division << "|" << address << endl;

file.close();

}

void deleteStudentInfo(int rollNumber) {

ifstream inFile("student\_info.txt");

ofstream outFile("temp.txt");

int r;

string line;

bool found = false;

while (getline(inFile, line)) {

stringstream ss(line);

ss >> r;

if (r != rollNumber)

outFile << line << endl;

else

found = true;

}

inFile.close();

outFile.close();

remove("student\_info.txt");

rename("temp.txt", "student\_info.txt");

if (!found)

cout << "Student with Roll Number " << rollNumber << " not found!" << endl;

else

cout << "Student with Roll Number " << rollNumber << " deleted successfully!" << endl;

}

void displayStudentInfo(int rollNumber) {

ifstream file("student\_info.txt");

if (!file.is\_open()) {

cout << "Error: Unable to open file!" << endl;

return;

}

int r;

string line;

bool found = false;

while (getline(file, line)) {

stringstream ss(line);

ss >> r;

if (r == rollNumber) {

found = true;

cout << line << endl;

break;

}

}

file.close();

if (!found)

cout << "Student with Roll Number " << rollNumber << " not found!" << endl;

}

int main() {

int choice, rollNumber;

char ch;

do {

cout << "\n1. Add Student Information\n2. Delete Student Information\n3. Display Student Information\n4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

addStudentInfo();

break;

case 2:

cout << "Enter Roll Number to delete: ";

cin >> rollNumber;

deleteStudentInfo(rollNumber);

break;

case 3:

cout << "Enter Roll Number to display: ";

cin >> rollNumber;

displayStudentInfo(rollNumber);

break;

case 4:

cout << "Exiting the program..." << endl;

break;

default:

cout << "Invalid choice!" << endl;

}

cout << "\nDo you want to continue (y/n)? ";

cin >> ch;

} while (ch == 'y' || ch == 'Y');

return 0;

}

OUTPUT : 1. Add Student Information

2. Delete Student Information

3. Display Student Information

4. Exit

Enter your choice: 1

Enter Roll Number: 27

Enter Name: PRANAV

Enter Division: SE

Enter Address: XYZ

Do you want to continue (y/n)? N

ASSIGNMENT 12:

Q. Company maintains employee information as employee ID, name, designation, and

salary. Allow user to add, delete information of employee. Display information of

particular employee. If employee does not exist an appropriate message is displayed.

If it is, then the system displays the employee details. Use index sequential file to

maintain the data.

CODE:

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <iomanip>

using namespace std;

struct Employee {

int id;

string name;

string designation;

double salary;

Employee(int \_id, const string& \_name, const string& \_designation, double \_salary) :

id(\_id), name(\_name), designation(\_designation), salary(\_salary) {}

};

void addEmployeeInfo(const Employee& emp) {

ofstream file("employee\_info.txt", ios::app);

if (!file.is\_open()) {

cout << "Error: Unable to open file!" << endl;

return;

}

file << emp.id << "|" << emp.name << "|" << emp.designation << "|" << emp.salary << endl;

file.close();

}

void deleteEmployeeInfo(int id) {

ifstream inFile("employee\_info.txt");

ofstream outFile("temp.txt");

int empId;

string line;

bool found = false;

while (getline(inFile, line)) {

stringstream ss(line);

ss >> empId;

if (empId != id)

outFile << line << endl;

else

found = true;

}

inFile.close();

outFile.close();

remove("employee\_info.txt");

rename("temp.txt", "employee\_info.txt");

if (!found)

cout << "Employee with ID " << id << " not found!" << endl;

else

cout << "Employee with ID " << id << " deleted successfully!" << endl;

}

void displayEmployeeInfo(int id) {

ifstream file("employee\_info.txt");

if (!file.is\_open()) {

cout << "Error: Unable to open file!" << endl;

return;

}

int empId;

string line;

bool found = false;

while (getline(file, line)) {

stringstream ss(line);

ss >> empId;

if (empId == id) {

found = true;

cout << line << endl;

break;

}

}

file.close();

if (!found)

cout << "Employee with ID " << id << " not found!" << endl;

}

int main() {

int choice, id;

char ch;

do {

cout << "\n1. Add Employee Information\n2. Delete Employee Information\n3. Display Employee Information\n4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int empId;

string name, designation;

double salary;

cout << "Enter Employee ID: ";

cin >> empId;

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // Clear input buffer

cout << "Enter Employee Name: ";

getline(cin, name);

cout << "Enter Employee Designation: ";

getline(cin, designation);

cout << "Enter Employee Salary: ";

cin >> salary;

Employee emp(empId, name, designation, salary);

addEmployeeInfo(emp);

break;

}

case 2:

cout << "Enter Employee ID to delete: ";

cin >> id;

deleteEmployeeInfo(id);

break;

case 3:

cout << "Enter Employee ID to display: ";

cin >> id;

displayEmployeeInfo(id);

break;

case 4:

cout << "Exiting the program..." << endl;

break;

default:

cout << "Invalid choice!" << endl;

}

cout << "\nDo you want to continue (y/n)? ";

cin >> ch;

} while (ch == 'y' || ch == 'Y');

return 0;

}

OUTPUT : 1. Add Employee Information

2. Delete Employee Information

3. Display Employee Information

4. Exit

Enter your choice: 1

Enter Employee ID: 31313

Enter Employee Name: pranav

Enter Employee Designation: manager

Enter Employee Salary: 50k

Do you want to continue (y/n)?

CLASS COORDINATER: BHARTI BISANE

STUDENT NAME: PRANAV KULAKRNI