**OOPS ASSIGNMENT 2**

**NAME: ABDUL HANNAN SHAHID**

**REG NO# : 864/BSIT/F22/FOC {A}**

The updated code is available on this link:  
https://github.com/ABDULHANNANSHAH/oopassignment2

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

string Name;

string Father\_Name;

int Emp\_id;

string dob;

int bps;

string job\_Type;

public:

Employee() {}

Employee(string name, string fatherName, int empId, string dateOfBirth, int bpsValue, string jobType)

: Name(name), Father\_Name(fatherName), Emp\_id(empId), dob(dateOfBirth), bps(bpsValue), job\_Type(jobType) {}

void setRecord() {

cout << "Enter Name: ";

cin >> Name;

cout << "Enter Father's Name: ";

cin >> Father\_Name;

cout << "Enter Employee ID: ";

cin >> Emp\_id;

cout << "Enter Date of Birth: ";

cin >> dob;

cout << "Enter BPS: ";

cin >> bps;

cout << "Enter Job Type: ";

cin >> job\_Type;

}

void showRecord() {

cout << "Name: " << Name << endl;

cout << "Father's Name: " << Father\_Name << endl;

cout << "Employee ID: " << Emp\_id << endl;

cout << "Date of Birth: " << dob << endl;

cout << "BPS: " << bps << endl;

cout << "Job Type: " << job\_Type << endl;

}

};

class Teacher : public Employee {

private:

string Education;

string Teaching\_level;

string Subject;

public:

Teacher() {}

Teacher(string name, string fatherName, int empId, string dateOfBirth, int bpsValue, string jobType,

string education, string teachingLevel, string subject)

: Employee(name, fatherName, empId, dateOfBirth, bpsValue, jobType),

Education(education), Teaching\_level(teachingLevel), Subject(subject) {}

void setRecord() {

Employee::setRecord();

cout << "Enter Education: ";

cin >> Education;

cout << "Enter Teaching Level: ";

cin >> Teaching\_level;

cout << "Enter Subject: ";

cin >> Subject;

}

void showRecord() {

Employee::showRecord();

cout << "Education: " << Education << endl;

cout << "Teaching Level: " << Teaching\_level << endl;

cout << "Subject: " << Subject << endl;

}

void upDateRecord() {

cout << "Enter new Job Type: ";

cin >> job\_Type;

cout << "Enter new BPS: ";

cin >> bps;

cout << "Enter new Education: ";

cin >> Education;

cout << "Enter new Teaching Level: ";

cin >> Teaching\_level;

}

};

class Doctor : public Employee {

private:

string Doctor\_type;

string Specialization;

string Job\_level;

public:

Doctor() {}

Doctor(string name, string fatherName, int empId, string dateOfBirth, int bpsValue, string jobType,

string doctorType, string specialization, string jobLevel)

: Employee(name, fatherName, empId, dateOfBirth, bpsValue, jobType),

Doctor\_type(doctorType), Specialization(specialization), Job\_level(jobLevel) {}

void setRecord() {

Employee::setRecord();

cout << "Enter Doctor Type: ";

cin >> Doctor\_type;

cout << "Enter Specialization: ";

cin >> Specialization;

cout << "Enter Job Level: ";

cin >> Job\_level;

}

void showRecord() {

Employee::showRecord();

cout << "Doctor Type: " << Doctor\_type << endl;

cout << "Specialization: " << Specialization << endl;

cout << "Job Level: " << Job\_level << endl;

}

void upDateRecord() {

cout << "Enter new Job Type: ";

cin >> job\_Type;

cout << "Enter new BPS: ";

cin >> bps;

cout << "Enter new Job Level: ";

cin >> Job\_level;

}

};

class TeacherNode {

public:

Teacher teacher;

TeacherNode\* next;

TeacherNode(Teacher t) : teacher(t), next(nullptr) {}

};

class DoctorNode {

public:

Doctor doctor;

DoctorNode\* next;

DoctorNode(Doctor d) : doctor(d), next(nullptr) {}

};

class EmployeeList {

private:

TeacherNode\* teacherHead;

DoctorNode\* doctorHead;

public:

EmployeeList() : teacherHead(nullptr), doctorHead(nullptr) {}

void AddNewDoctor() {

Doctor d;

d.setRecord();

DoctorNode\* newNode = new DoctorNode(d);

if (doctorHead == nullptr) {

doctorHead = newNode;

} else {

DoctorNode\* temp = doctorHead;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

void AddNewTeacher() {

Teacher t;

t.setRecord();

TeacherNode\* newNode = new TeacherNode(t);

if (teacherHead == nullptr) {

teacherHead = newNode;

} else {

TeacherNode\* temp = teacherHead;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

void updateEmployeeRecord(int empId) {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Emp\_id == empId) {

teacherTemp->teacher.upDateRecord();

return;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Emp\_id == empId) {

doctorTemp->doctor.upDateRecord();

return;

}

doctorTemp = doctorTemp->next;

}

cout << "Employee with ID " << empId << " not found." << endl;

}

void SearchEmployeeByID(int empId) {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Emp\_id == empId) {

teacherTemp->teacher.showRecord();

return;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Emp\_id == empId) {

doctorTemp->doctor.showRecord();

return;

}

doctorTemp = doctorTemp->next;

}

cout << "Employee with ID " << empId << " not found." << endl;

}

void SearchEmployeeByName(string name) {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

bool found = false;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Name == name) {

teacherTemp->teacher.showRecord();

found = true;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Name == name) {

doctorTemp->doctor.showRecord();

found = true;

}

doctorTemp = doctorTemp->next;

}

if (!found) {

cout << "Employee with name " << name << " not found." << endl;

}

}

void SearchYoungestEmployee() {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

int minAge = INT\_MAX;

string youngestEmployee;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.dob < minAge) {

minAge = teacherTemp->teacher.dob;

youngestEmployee = teacherTemp->teacher.Name;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.dob < minAge) {

minAge = doctorTemp->doctor.dob;

youngestEmployee = doctorTemp->doctor.Name;

}

doctorTemp = doctorTemp->next;

}

cout << "Youngest Employee: " << youngestEmployee << endl;

}

void SearchEldestEmployee() {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

int maxAge = INT\_MIN;

string eldestEmployee;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.dob > maxAge) {

maxAge = teacherTemp->teacher.dob;

eldestEmployee = teacherTemp->teacher.Name;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.dob > maxAge) {

maxAge = doctorTemp->doctor.dob;

eldestEmployee = doctorTemp->doctor.Name;

}

doctorTemp = doctorTemp->next;

}

cout << "Eldest Employee: " << eldestEmployee << endl;

}

void SearchDoctorBySpecialization(string specialization) {

DoctorNode\* doctorTemp = doctorHead;

bool found = false;

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Specialization == specialization) {

doctorTemp->doctor.showRecord();

found = true;

}

doctorTemp = doctorTemp->next;

}

if (!found) {

cout << "No doctors found with specialization " << specialization << endl;

}

}

void SearchTeacherByTeachingLevel(string teachingLevel) {

TeacherNode\* teacherTemp = teacherHead;

bool found = false;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Teaching\_level == teachingLevel) {

teacherTemp->teacher.showRecord();

found = true;

}

teacherTemp = teacherTemp->next;

}

if (!found) {

cout << "No teachers found with teaching level " << teachingLevel << endl;

}

}

void DeleteEmployeeByID(int empId) {

TeacherNode\* teacherTemp = teacherHead;

TeacherNode\* teacherPrev = nullptr;

DoctorNode\* doctorTemp = doctorHead;

DoctorNode\* doctorPrev = nullptr;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Emp\_id == empId) {

if (teacherPrev == nullptr) {

teacherHead = teacherTemp->next;

} else {

teacherPrev->next = teacherTemp->next;

}

delete teacherTemp;

cout << "Employee with ID " << empId << " deleted." << endl;

return;

}

teacherPrev = teacherTemp;

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Emp\_id == empId) {

if (doctorPrev == nullptr) {

doctorHead = doctorTemp->next;

} else {

doctorPrev->next = doctorTemp->next;

}

delete doctorTemp;

cout << "Employee with ID " << empId << " deleted." << endl;

return;

}

doctorPrev = doctorTemp;

doctorTemp = doctorTemp->next;

}

cout << "Employee with ID " << empId << " not found." << endl;

}

void DeleteEmployeeByName(string name) {

TeacherNode\* teacherTemp = teacherHead;

TeacherNode\* teacherPrev = nullptr;

DoctorNode\* doctorTemp = doctorHead;

DoctorNode\* doctorPrev = nullptr;

bool found = false;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Name == name) {

if (teacherPrev == nullptr) {

teacherHead = teacherTemp->next;

} else {

teacherPrev->next = teacherTemp->next;

}

delete teacherTemp;

found = true;

}

teacherPrev = teacherTemp;

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Name == name) {

if (doctorPrev == nullptr) {

doctorHead = doctorTemp->next;

} else {

doctorPrev->next = doctorTemp->next;

}

delete doctorTemp;

found = true;

}

doctorPrev = doctorTemp;

doctorTemp = doctorTemp->next;

}

if (found) {

cout << "Employees with name " << name << " deleted." << endl;

} else {

cout << "No employees found with name " << name << endl;

}

}

};

int main() {

EmployeeList empList;

int choice;

do {

cout << "Menu:" << endl;

cout << "1. Add Employee" << endl;

cout << "2. Update Employee Record" << endl;

cout << "3. Search Employee" << endl;

cout << "4. Delete Employee" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int empType;

cout << "Add Employee:" << endl;

cout << "1. Doctor" << endl;

cout << "2. Teacher" << endl;

cout << "Enter employee type: ";

cin >> empType;

if (empType == 1) {

empList.AddNewDoctor();

} else if (empType == 2) {

empList.AddNewTeacher();

} else {

cout << "Invalid employee type." << endl;

}

break;

}

case 2: {

int empId;

cout << "Update Employee Record:" << endl;

cout << "Enter employee ID: ";

cin >> empId;

empList.updateEmployeeRecord(empId);

break;

}

case 3: {

int searchChoice;

cout << "Search Employee:" << endl;

cout << "1. Search by ID" << endl;

cout << "2. Search by Name" << endl;

cout << "3. Search Youngest Employee" << endl;

cout << "4. Search Eldest Employee" << endl;

cout << "5. Search Doctor by Specialization" << endl;

cout << "6. Search Teacher by Teaching Level" << endl;

cout << "Enter search choice: ";

cin >> searchChoice;

if (searchChoice == 1) {

int empId;

cout << "Enter employee ID: ";

cin >> empId;

empList.SearchEmployeeByID(empId);

} else if (searchChoice == 2) {

string name;

cout << "Enter employee name: ";

cin >> name;

empList.SearchEmployeeByName(name);

} else if (searchChoice == 3) {

empList.SearchYoungestEmployee();

} else if (searchChoice == 4) {

empList.SearchEldestEmployee();

} else if (searchChoice == 5) {

string specialization;

cout << "Enter specialization: ";

cin >> specialization;

empList.SearchDoctorBySpecialization(specialization);

} else if (searchChoice == 6) {

string teachingLevel;

cout << "Enter teaching level: ";

cin >> teachingLevel;

empList.SearchTeacherByTeachingLevel(teachingLevel);

} else {

cout << "Invalid search choice." << endl;

}

break;

}

case 4: {

int deleteChoice;

cout << "Delete Employee:" << endl;

cout << "1. Delete by ID" << endl;

cout << "2. Delete by Name" << endl;

cout << "Enter delete choice: ";

cin >> deleteChoice;

if (deleteChoice == 1) {

int empId;

cout << "Enter employee ID: ";

cin >> empId;

empList.DeleteEmployeeByID(empId);

} else if (deleteChoice == 2) {

string name;

cout << "Enter employee name: ";

cin >> name;

empList.DeleteEmployeeByName(name);

} else {

cout << "Invalid delete choice." << endl;

}

break;

}

case 0:

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

break;

default:

cout << "Invalid choice." << endl;

break;

}

cout << endl;

} while (choice != 0);

return 0;

}

Second code:  
#include <iostream>

#include <string>

#include <climits>

using namespace std;

class Employee {

private:

string Name;

string Father\_Name;

int Emp\_id;

string dob;

int bps;

string job\_Type;

public:

Employee() {}

Employee(string name, string fatherName, int empId, string dateOfBirth, int bpsValue, string jobType)

: Name(name), Father\_Name(fatherName), Emp\_id(empId), dob(dateOfBirth), bps(bpsValue), job\_Type(jobType) {}

void setRecord() {

cout << "Enter Name: ";

cin >> Name;

cout << "Enter Father's Name: ";

cin >> Father\_Name;

cout << "Enter Employee ID: ";

cin >> Emp\_id;

cout << "Enter Date of Birth: ";

cin >> dob;

cout << "Enter BPS: ";

cin >> bps;

cout << "Enter Job Type: ";

cin >> job\_Type;

}

void showRecord() {

cout << "Name: " << Name << endl;

cout << "Father's Name: " << Father\_Name << endl;

cout << "Employee ID: " << Emp\_id << endl;

cout << "Date of Birth: " << dob << endl;

cout << "BPS: " << bps << endl;

cout << "Job Type: " << job\_Type << endl;

}

};

class Teacher : public Employee {

private:

string Education;

string Teaching\_level;

string Subject;

public:

Teacher() {}

Teacher(string name, string fatherName, int empId, string dateOfBirth, int bpsValue, string jobType,

string education, string teachingLevel, string subject)

: Employee(name, fatherName, empId, dateOfBirth, bpsValue, jobType),

Education(education), Teaching\_level(teachingLevel), Subject(subject) {}

void setRecord() {

Employee::setRecord();

cout << "Enter Education: ";

cin >> Education;

cout << "Enter Teaching Level: ";

cin >> Teaching\_level;

cout << "Enter Subject: ";

cin >> Subject;

}

void showRecord() {

Employee::showRecord();

cout << "Education: " << Education << endl;

cout << "Teaching Level: " << Teaching\_level << endl;

cout << "Subject: " << Subject << endl;

}

void updateRecord() {

cout << "Enter new Job Type: ";

cin >> job\_Type;

cout << "Enter new BPS: ";

cin >> bps;

cout << "Enter new Education: ";

cin >> Education;

cout << "Enter new Teaching Level: ";

cin >> Teaching\_level;

}

};

class Doctor : public Employee {

private:

string Doctor\_type;

string Specialization;

string Job\_level;

public:

Doctor() {}

Doctor(string name, string fatherName, int empId, string dateOfBirth, int bpsValue, string jobType,

string doctorType, string specialization, string jobLevel)

: Employee(name, fatherName, empId, dateOfBirth, bpsValue, jobType),

Doctor\_type(doctorType), Specialization(specialization), Job\_level(jobLevel) {}

void setRecord() {

Employee::setRecord();

cout << "Enter Doctor Type: ";

cin >> Doctor\_type;

cout << "Enter Specialization: ";

cin >> Specialization;

cout << "Enter Job Level: ";

cin >> Job\_level;

}

void showRecord() {

Employee::showRecord();

cout << "Doctor Type: " << Doctor\_type << endl;

cout << "Specialization: " << Specialization << endl;

cout << "Job Level: " << Job\_level << endl;

}

void updateRecord() {

cout << "Enter new Job Type: ";

cin >> job\_Type;

cout << "Enter new BPS: ";

cin >> bps;

cout << "Enter new Job Level: ";

cin >> Job\_level;

}

};

class TeacherNode {

public:

Teacher teacher;

TeacherNode\* next;

TeacherNode(Teacher t) : teacher(t), next(nullptr) {}

};

class DoctorNode {

public:

Doctor doctor;

DoctorNode\* next;

DoctorNode(Doctor d) : doctor(d), next(nullptr) {}

};

class EmployeeList {

private:

TeacherNode\* teacherHead;

DoctorNode\* doctorHead;

public:

EmployeeList() : teacherHead(nullptr), doctorHead(nullptr) {}

void AddNewDoctor() {

Doctor d;

d.setRecord();

DoctorNode\* newNode = new DoctorNode(d);

if (doctorHead == nullptr) {

doctorHead = newNode;

} else {

DoctorNode\* temp = doctorHead;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

void AddNewTeacher() {

Teacher t;

t.setRecord();

TeacherNode\* newNode = new TeacherNode(t);

if (teacherHead == nullptr) {

teacherHead = newNode;

} else {

TeacherNode\* temp = teacherHead;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

void updateEmployeeRecord(int empId) {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Emp\_id == empId) {

teacherTemp->teacher.updateRecord();

return;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Emp\_id == empId) {

doctorTemp->doctor.updateRecord();

return;

}

doctorTemp = doctorTemp->next;

}

cout << "Employee with ID " << empId << " not found." << endl;

}

void SearchEmployeeByID(int empId) {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Emp\_id == empId) {

teacherTemp->teacher.showRecord();

return;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Emp\_id == empId) {

doctorTemp->doctor.showRecord();

return;

}

doctorTemp = doctorTemp->next;

}

cout << "Employee with ID " << empId << " not found." << endl;

}

void SearchEmployeeByName(string name) {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

bool found = false;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Name == name) {

teacherTemp->teacher.showRecord();

found = true;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Name == name) {

doctorTemp->doctor.showRecord();

found = true;

}

doctorTemp = doctorTemp->next;

}

if (!found) {

cout << "Employee with name " << name << " not found." << endl;

}

}

void SearchYoungestEmployee() {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

string youngestEmployee;

int minAge = INT\_MAX;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.dob < minAge) {

minAge = teacherTemp->teacher.dob;

youngestEmployee = teacherTemp->teacher.Name;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.dob < minAge) {

minAge = doctorTemp->doctor.dob;

youngestEmployee = doctorTemp->doctor.Name;

}

doctorTemp = doctorTemp->next;

}

cout << "Youngest Employee: " << youngestEmployee << endl;

}

void SearchEldestEmployee() {

TeacherNode\* teacherTemp = teacherHead;

DoctorNode\* doctorTemp = doctorHead;

string eldestEmployee;

int maxAge = INT\_MIN;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.dob > maxAge) {

maxAge = teacherTemp->teacher.dob;

eldestEmployee = teacherTemp->teacher.Name;

}

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.dob > maxAge) {

maxAge = doctorTemp->doctor.dob;

eldestEmployee = doctorTemp->doctor.Name;

}

doctorTemp = doctorTemp->next;

}

cout << "Eldest Employee: " << eldestEmployee << endl;

}

void SearchDoctorBySpecialization(string specialization) {

DoctorNode\* doctorTemp = doctorHead;

bool found = false;

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Specialization == specialization) {

doctorTemp->doctor.showRecord();

found = true;

}

doctorTemp = doctorTemp->next;

}

if (!found) {

cout << "No doctors found with specialization " << specialization << endl;

}

}

void SearchTeacherByTeachingLevel(string teachingLevel) {

TeacherNode\* teacherTemp = teacherHead;

bool found = false;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Teaching\_level == teachingLevel) {

teacherTemp->teacher.showRecord();

found = true;

}

teacherTemp = teacherTemp->next;

}

if (!found) {

cout << "No teachers found with teaching level " << teachingLevel << endl;

}

}

void DeleteEmployeeByID(int empId) {

TeacherNode\* teacherTemp = teacherHead;

TeacherNode\* teacherPrev = nullptr;

DoctorNode\* doctorTemp = doctorHead;

DoctorNode\* doctorPrev = nullptr;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Emp\_id == empId) {

if (teacherPrev == nullptr) {

teacherHead = teacherTemp->next;

} else {

teacherPrev->next = teacherTemp->next;

}

delete teacherTemp;

cout << "Employee with ID " << empId << " deleted." << endl;

return;

}

teacherPrev = teacherTemp;

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Emp\_id == empId) {

if (doctorPrev == nullptr) {

doctorHead = doctorTemp->next;

} else {

doctorPrev->next = doctorTemp->next;

}

delete doctorTemp;

cout << "Employee with ID " << empId << " deleted." << endl;

return;

}

doctorPrev = doctorTemp;

doctorTemp = doctorTemp->next;

}

cout << "Employee with ID " << empId << " not found." << endl;

}

void DeleteEmployeeByName(string name) {

TeacherNode\* teacherTemp = teacherHead;

TeacherNode\* teacherPrev = nullptr;

DoctorNode\* doctorTemp = doctorHead;

DoctorNode\* doctorPrev = nullptr;

bool found = false;

while (teacherTemp != nullptr) {

if (teacherTemp->teacher.Name == name) {

if (teacherPrev == nullptr) {

teacherHead = teacherTemp->next;

} else {

teacherPrev->next = teacherTemp->next;

}

delete teacherTemp;

found = true;

}

teacherPrev = teacherTemp;

teacherTemp = teacherTemp->next;

}

while (doctorTemp != nullptr) {

if (doctorTemp->doctor.Name == name) {

if (doctorPrev == nullptr) {

doctorHead = doctorTemp->next;

} else {

doctorPrev->next = doctorTemp->next;

}

delete doctorTemp;

found = true;

}

doctorPrev = doctorTemp;

doctorTemp = doctorTemp->next;

}

if (found) {

cout << "Employees with name " << name << " deleted." << endl;

} else {

cout << "No employees found with name " << name << endl;

}

}

};

int main() {

EmployeeList empList;

int choice;

do {

cout << "Menu:" << endl;

cout << "1. Add Employee" << endl;

cout << "2. Update Employee Record" << endl;

cout << "3. Search Employee" << endl;

cout << "4. Delete Employee" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int empType;

cout << "Add Employee:" << endl;

cout << "1. Doctor" << endl;

cout << "2. Teacher" << endl;

cout << "Enter employee type: ";

cin >> empType;

if (empType == 1) {

empList.AddNewDoctor();

} else if (empType == 2) {

empList.AddNewTeacher();

} else {

cout << "Invalid employee type." << endl;

}

break;

}

case 2: {

int empId;

cout << "Update Employee Record:" << endl;

cout << "Enter employee ID: ";

cin >> empId;

empList.updateEmployeeRecord(empId);

break;

}

case 3: {

int searchChoice;

cout << "Search Employee:" << endl;

cout << "1. Search by ID" << endl;

cout << "2. Search by Name" << endl;

cout << "3. Search Youngest Employee" << endl;

cout << "4. Search Eldest Employee" << endl;

cout << "5. Search Doctor by Specialization" << endl;

cout << "6. Search Teacher by Teaching Level" << endl;

cout << "Enter search choice: ";

cin >> searchChoice;

if (searchChoice == 1) {

int empId;

cout << "Enter employee ID: ";

cin >> empId;

empList.SearchEmployeeByID(empId);

} else if (searchChoice == 2) {

string name;

cout << "Enter employee name: ";

cin >> name;

empList.SearchEmployeeByName(name);

} else if (searchChoice == 3) {

empList.SearchYoungestEmployee();

} else if (searchChoice == 4) {

empList.SearchEldestEmployee();

} else if (searchChoice == 5) {

string specialization;

cout << "Enter specialization: ";

cin >> specialization;

empList.SearchDoctorBySpecialization(specialization);

} else if (searchChoice == 6) {

string teachingLevel;

cout << "Enter teaching level: ";

cin >> teachingLevel;

empList.SearchTeacherByTeachingLevel(teachingLevel);

} else {

cout << "Invalid search choice." << endl;

}

break;

}

case 4: {

int deleteChoice;

cout << "Delete Employee:" << endl;

cout << "1. Delete by ID" << endl;

cout << "2. Delete by Name" << endl;

cout << "Enter delete choice: ";

cin >> deleteChoice;

if (deleteChoice == 1) {

int empId;

cout << "Enter employee ID: ";

cin >> empId;

empList.DeleteEmployeeByID(empId);

} else if (deleteChoice == 2) {

string name;

cout << "Enter employee name: ";

cin >> name;

empList.DeleteEmployeeByName(name);

} else {

cout << "Invalid delete choice." << endl;

}

break;

}

case 0:

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

break;

default:

cout << "Invalid choice." << endl;

break;

}

cout << endl;

} while (choice != 0);

return 0;

}