**Institute of Computer Technology**

**B. Tech. Computer Science and Engineering**

**Sub: ESFP – II**

**Course Code: 2CSE203**

**Practical – 10**

**Name: Jaymin Gondaliya**

**Enrollment No: 23162171007**

**Sem - 2**

**Branch: CS**

**Class: B**

**Batch: 25**

**Objective:**

To learn about object-oriented features, polymorphism (Function & Operator overloading).

**Problem Definition-1:** Complete the code for the object assigned to you to satisfy following specifications.

**Code:**

#include <iostream>

#include <string>

using namespace std;

const int MAX\_EMPLOYEES = 100;

class EmployeeManager;

class EmployeeDatabase;

class Employee

{

    friend class EmployeeManager;

    friend class EmployeeDatabase;

private:

    string name;

    int id;

    double salary;

    bool isPresent;

    string department;

public:

    Employee() : isPresent(false) {}

    ~Employee()

    {

        cout << "Destroying employee: " << name << endl;

    }

    void markAttendance(bool present)

    {

        isPresent = present;

        if (!isPresent)

        {

            salary -= 50;

        }

    }

    void resetSalary()

    {

        salary = 0;

    }

    void increaseSalary(double amount)

    {

        salary += amount;

    }

    bool operator>(const Employee &emp)

    {

        return name > emp.name;

    }

    bool operator<(const Employee &emp)

    {

        return id < emp.id;

    }

    bool operator==(const Employee &emp)

    {

        return salary == emp.salary;

    }

    friend ostream &operator<<(ostream &out, const Employee &emp)

    {

        out << "Name: " << emp.name << endl;

        out << "ID: " << emp.id << endl;

        out << "Salary: " << emp.salary << endl;

        out << "Department: " << emp.department << endl;

        return out;

    }

    friend istream &operator>>(istream &in, Employee &emp)

    {

        cout << "Enter employee name: ";

        in >> emp.name;

        cout << "Enter employee ID: ";

        in >> emp.id;

        cout << "Enter employee salary: ";

        in >> emp.salary;

        cout << "Enter employee department: ";

        in >> emp.department;

        return in;

    }

    void setDepartment(const string &dept)

    {

        department = dept;

    }

    string getAttendanceStatus() const

    {

        return isPresent ? "Present" : "Absent";

    }

};

class EmployeeManager

{

private:

    Employee employees[MAX\_EMPLOYEES];

    int numEmployees;

public:

    EmployeeManager() : numEmployees(0) {}

    ~EmployeeManager()

    {

        cout << "Destroying EmployeeManager..." << endl;

    }

    int getNumEmployees() const

    {

        return numEmployees;

    }

    const Employee &getEmployee(int index) const

    {

        return employees[index];

    }

    void addEmployee(const Employee &emp)

    {

        if (numEmployees < MAX\_EMPLOYEES)

        {

            employees[numEmployees] = emp;

            numEmployees++;

        }

        else

        {

            cout << "Maximum number of employees reached." << endl;

        }

    }

    void markAttendance(int id, bool present)

    {

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

        {

            if (employees[i].id == id)

            {

                employees[i].markAttendance(present);

                cout << "Attendance marked for employee with ID " << id << endl;

                return;

            }

        }

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

    }

    void resetSalary(int id)

    {

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

        {

            if (employees[i].id == id)

            {

                employees[i].resetSalary();

                cout << "Salary reset for employee with ID " << id << endl;

                return;

            }

        }

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

    }

    void increaseSalary(int id, double amount)

    {

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

        {

            if (employees[i].id == id)

            {

                employees[i].increaseSalary(amount);

                cout << "Salary increased for employee with ID " << id << endl;

                return;

            }

        }

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

    }

};

class EmployeeDatabase

{

public:

    void displayByCategory(const EmployeeManager &empManager)

    {

        int choice;

        cout << "Display employees by category:" << endl;

        cout << "1. IT" << endl;

        cout << "2. HR" << endl;

        cout << "3. Finance" << endl;

        cout << "Enter choice: ";

        cin >> choice;

        cout << "Employees:" << endl;

        for (int i = 0; i < empManager.getNumEmployees(); ++i)

        {

            const Employee &employee = empManager.getEmployee(i);

            if (choice == 1 && employee.department == "IT")

            {

                cout << employee << "Attendance: " << employee.getAttendanceStatus() << endl;

            }

            else if (choice == 2 && employee.department == "HR")

            {

                cout << employee << "Attendance: " << employee.getAttendanceStatus() << endl;

            }

            else if (choice == 3 && employee.department == "Finance")

            {

                cout << employee << "Attendance: " << employee.getAttendanceStatus() << endl;

            }

        }

    }

};

int main()

{

    EmployeeManager empManager;

    EmployeeDatabase empDatabase;

    int choice;

    do

    {

        cout << "\nEmployee Management System\n";

        cout << "1. Enter Employee Details\n";

        cout << "2. Display All Employees\n";

        cout << "3. Mark Attendance\n";

        cout << "4. Reset Salary\n";

        cout << "5. Increase Salary\n";

        cout << "6. Display Employees by Category\n";

        cout << "7. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            Employee emp;

            cin >> emp;

            empManager.addEmployee(emp);

            break;

        }

        case 2:

        {

            cout << "\nEmployee Details:" << endl;

            for (int i = 0; i < empManager.getNumEmployees(); ++i)

            {

                cout << empManager.getEmployee(i) << "Attendance: " << empManager.getEmployee(i).getAttendanceStatus() << endl;

            }

            break;

        }

        case 3:

        {

            int employeeId;

            bool present;

            cout << "\nEnter employee ID to mark attendance: ";

            cin >> employeeId;

            cout << "Enter 1 for present, 0 for absent: ";

            cin >> present;

            empManager.markAttendance(employeeId, present);

            break;

        }

        case 4:

        {

            int employeeId;

            cout << "\nEnter employee ID to reset salary: ";

            cin >> employeeId;

            empManager.resetSalary(employeeId);

            break;

        }

        case 5:

        {

            int employeeId;

            double amount;

            cout << "\nEnter employee ID to increase salary: ";

            cin >> employeeId;

            cout << "Enter amount to increase: ";

            cin >> amount;

            empManager.increaseSalary(employeeId, amount);

            break;

        }

        case 6:

        {

            empDatabase.displayByCategory(empManager);

            break;

        }

        case 7:

        {

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

            break;

        }

        default:

            cout << "Invalid choice. Please try again." << endl;

        }

    } while (choice != 7);

    return 0;

}

**Output –**

****