**COMPUTER SCIENCE DEPARTMENT**

Object Oriented Programming

**Assignment # 04**

**Last date of Submission: 19th May 2024**

# Submitted To: MAM Najla Raza

**Student Name:**

# Reg. Number:

# 

**Question no.1**

Analyze the scenario of managing student records stored in a text file, calculating average grades, and storing results in another file. Identify potential challenges such as data inconsistencies or file-related errors. Develop a solution that involves reading and processing the student records, calculating average grades, and storing results while incorporating robust error-handling mechanisms to address potential issues and ensure the reliability of the program. Implement solution using C++.

**Code**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Student {

public:

    string name;

    int rollno;

    int marks[5];

    char grade;

    Student() {

        name = "";

        rollno = 0;

        for (int i = 0; i < 5; ++i) {

            marks[i] = 0;

        }

        grade = 'F';

    }

    void calculateGrade() {

        int sum = 0;

        for (int i = 0; i < 5; ++i) {

            sum += marks[i];

        }

        double average = sum / 5.0;

        if (average >= 90) grade = 'A';

        else if (average >= 80) grade = 'B';

        else if (average >= 70) grade = 'C';

        else if (average >= 60) grade = 'D';

        else grade = 'F';

    }

    void display() const {

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

        cout << "Roll No: " << rollno << endl;

        cout << "Marks: ";

        for (int i = 0; i < 5; ++i) {

            cout << marks[i] << " ";

        }

        cout << endl;

        cout << "Grade: " << grade << endl << endl;

    }

};

void saveNameAndGrade(const string& *filename*, const string& *name*, char *grade*) {

    ofstream outfile(*filename*, ios::app);

    if (!outfile.is\_open()) {

        cout << "Failed to open file for writing: " << *filename* << endl;

        return;

    }

    outfile << *name* << ", " << *grade* << endl;

    outfile.close();

}

int main() {

    ifstream file("data.txt");

    if (!file.is\_open()) {

        cout << "Could not load data from file" << endl;

        cout << "Exiting" << endl;

        return 0;

    }

    ofstream outFile("grades.txt", ios::trunc); // Open the file in trunc mode to clear existing content

    outFile.close();

    while (!file.eof()) {

        Student s1;

        getline(file, s1.name);

        if (s1.name.empty()) {

            continue; // Skip any empty lines

        }

        file >> s1.rollno;

        for (int i = 0; i < 5; ++i) {

            file >> s1.marks[i];

        }

        file.ignore(); // Ignore the newline character after reading marks

        s1.calculateGrade();

        s1.display();

        saveNameAndGrade("grades.txt", s1.name, s1.grade);

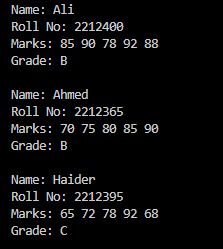
    }

    file.close();

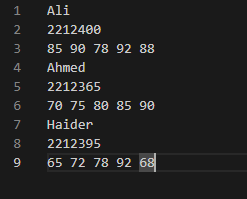
    return 0;

}

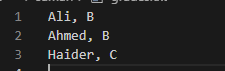
**Output**



**Data.txt**



**grades.txt**



**Question no.2**

**Code**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

// Base class for all employees

class Employee {

protected:

    string name;

    int id;

public:

    Employee(const string& *name*, int *id*) : name(*name*), id(*id*) {}

    virtual double calculateSalary() const = 0;

    virtual void display() const {

        cout << "Name: " << name << ", ID: " << id;

    }

    virtual ~Employee() = default;

    string getName() const { return name; }

    int getId() const { return id; }

};

// Derived class for full-time employees

class FullTimeEmployee : public Employee {

private:

    double monthlySalary;

public:

    FullTimeEmployee(const string& *name*, int *id*, double *salary*)

        : Employee(*name*, *id*), monthlySalary(*salary*) {}

    double calculateSalary() const override {

        return monthlySalary;

    }

    void display() const override {

        Employee::display();

        cout << ", Monthly Salary: " << monthlySalary << endl;

    }

};

// Derived class for part-time employees

class PartTimeEmployee : public Employee {

private:

    double hourlyRate;

    int hoursWorked;

public:

    PartTimeEmployee(const string& *name*, int *id*, double *rate*, int *hours*)

        : Employee(*name*, *id*), hourlyRate(*rate*), hoursWorked(*hours*) {}

    double calculateSalary() const override {

        return hourlyRate \* hoursWorked;

    }

    void display() const override {

        Employee::display();

        cout << ", Hourly Rate: " << hourlyRate << ", Hours Worked: " << hoursWorked << endl;

    }

};

// Payroll Management System

class PayrollSystem {

private:

    static const int MAX\_EMPLOYEES = 100;

    Employee\* employees[MAX\_EMPLOYEES];

    int employeeCount;

public:

    PayrollSystem() : employeeCount(0) {}

    void addEmployee(Employee\* *employee*) {

        if (employeeCount < MAX\_EMPLOYEES) {

            employees[employeeCount++] = *employee*;

        } else {

            cerr << "Cannot add more employees, maximum capacity reached." << endl;

        }

    }

    void calculateSalaries(const string& *outputFilename*) {

        ofstream outFile(*outputFilename*);

        if (!outFile.is\_open()) {

            cerr << "Could not open file for writing: " << *outputFilename* << endl;

            return;

        }

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

            outFile << "Name: " << employees[i]->getName() << ", ID: " << employees[i]->getId() << ", Salary: " << employees[i]->calculateSalary() << endl;

        }

        outFile.close();

    }

    ~PayrollSystem() {

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

            delete employees[i];

        }

    }

    void displayAllEmployees() const {

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

            employees[i]->display();

        }

    }

};

int main() {

    PayrollSystem payrollSystem;

    // Example employees

    payrollSystem.addEmployee(new FullTimeEmployee("Hassan", 1, 3000.0));

    payrollSystem.addEmployee(new PartTimeEmployee("Mir Hamza", 2, 20.0, 100));

    // Display all employees

    payrollSystem.displayAllEmployees();

    // Calculate salaries and store in file

    payrollSystem.calculateSalaries("salaries.txt");

    return 0;

}

**Output**



**salaries.txt**

