Batch: B1 Roll No.: 16010124080

Experiment / assignment / tutorial No. 3

TITLE : Student Result Processing System Using Classes in C++

AIM: To implement a student result processing system in C++ using classes and objects, focusing on encapsulation, constructors, and member functions

Expected OUTCOME of Experiment:

CO1:Apply the features of object oriented programming languages. (C++ and Java)

CO2:Explore arrays, vectors, classes and objects in C++ and Java

Books/ Journals/ Websites referred:

1. E. Balagurusamy, "Programming with Java", McGraw-Hill.
2. E. Balagurusamy, "Object Oriented Programming with C++", McGraw-Hill.

Design a C++ program that creates and manages records of students using classes. Your program must:

1. **Store data for multiple students.**
2. **Accept marks for 3 subjects.**
3. **Calculate total and average marks.**
4. **Display details of all students.**

5. Identify the student with the highest average.

Each student should have:

- **Roll Number (int)**
- **Name (string)**
- **Marks in 3 subjects (float)**
- **Total (float)**
- **Average (float)**

Requirements:

- **Create a class `Student` with private data members.**
- **Use public member functions to:**
  - ○ **Accept input (with a constructor or separate `input()` function).**
  - ○ **Calculate total and average.**
  - ○ **Display individual student data.**
- **Maintain multiple student records using an array of objects.**

Variations/Modifications:

1. Modify your class to include student grades (A/B/C) based on average marks. 2. Write a function to sort the students by total marks in descending order. 3. Rewrite the program using dynamic memory allocation instead of a fixed-size array.
4. Add a static data member to count how many Student objects were created. 5. Add a search function to find and display the record of a student by roll number.

Pre Lab/ Prior Concepts:

1. Basic Structure of a C++ Program

- Understanding the syntax of a C++ program: `#include`, `main()` function, return types.
- Use of headers like `<iostream>` and `using namespace std`.

2. Input and Output Operations

- Use of `cin` and `cout` for reading user input and displaying output. ● Formatting output using manipulators like `setw`, `fixed`, `setprecision` (if included).

3. Variables and Data Types

- Declaring variables of type `int`, `float`, `char`, and `string`.
- Understanding literals, constants, and data ranges.

4. Conditional Statements

- Use of `if`, `if-else`, and `switch` statements to perform decision-making based on input data.

5. Loops

- Implementation of `for`, `while`, and `do-while` loops.
- Use cases: repeating input operations, iterating over arrays or records.

6. Arrays

- Declaring and using single-dimensional arrays.
- Storing and accessing data for multiple entities like marks of students.

7. Functions

- Writing reusable blocks of code using functions.
- Function declaration, definition, calling, and parameter passing (by value/reference).

8. Introduction to Classes in C++

- Definition and declaration of a class.
- Understanding the syntax for:
    - **Private** and **Public** access specifiers.
    - **Member variables** and **member functions**.

- Creating **objects** and accessing class members.

9. Constructors (Optional for your experiment)

- Understanding how constructors are used to initialize object data automatically.
- Syntax of default and parameterized constructors.

10. Arrays of Objects

- Using an array to store multiple objects of a class (e.g., `Student s[50];`).
- Accessing member functions for each object inside a loop.

Algorithm:

Implementation details:

NO MODIFICATION

```
#include <iostream>

#include <string>

using namespace std;


class Student {

private:

 int rollNumber;

 string name;

 float marks[3];

 float total;

 float average;
```

```
public:
 Student() {
 total = 0;
 average = 0;
 }

 void input() {
 cout << "Enter Roll Number: ";
 cin >> rollNumber;
 cin.ignore();
 cout << "Enter Name: ";
 getline(cin, name);
 total = 0;
 for (int i = 0; i < 3; i++) {
 cout << "Enter marks for subject " << (i + 1) << ": ";
 cin >> marks[i];
 total += marks[i];
 }

 average = total / 3;

 }


 void calculateTotalAndAverage() {

 total = 0;

 for (int i = 0; i < 3; i++) {

 total += marks[i];

 }

 average = total / 3;

 }


 void display() const {

 cout << "Roll Number: " << rollNumber << "\n";

 cout << "Name: " << name << "\n";

 for (int i = 0; i < 3; i++) {

 cout << "Marks in subject " << (i + 1) << ": " << marks[i] << "\n";  }
```

```
 cout << "Total: " << total << "\n";

 cout << "Average: " << average << "\n";

 }


 float getAverage() const {

 return average;

 }

};


int findHighestAverage(Student* students, int n) {

 int idx = 0;

 float highestAvg = students[0].getAverage();

 for (int i = 1; i < n; i++) {

 if (students[i].getAverage() > highestAvg) {

 highestAvg = students[i].getAverage();

 idx = i;

 }

 }
```

```
 return idx;

}


int main() {

 int n;

 cout << "Enter number of students: ";

 cin >> n;

 Student* students = new Student[n];

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

 cout << "\nEnter details for student " << (i + 1) << ":\n";

students[i].input();

 }

 cout << "\nAll Student Details:\n";

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

 students[i].display();

 }

 int highestIdx = findHighestAverage(students, n);

 cout << "\nStudent with highest average:\n";
 students[highestIdx].display();

 delete[] students;

 return 0;

}
```

MODIFICATIONS

```
#include <iostream>

#include <string>

#include <vector>

using namespace std;


class Student {

private:

 int rollNumber;

 string name;

 float marks[3];

 float total;
 float average;
 char grade;

 static int studentCount;

 void calculateGrade() {
 if (average >= 90) grade = 'A';
 else if (average >= 75) grade = 'B';
 else if (average >= 60) grade = 'C';
 else grade = 'F';
 }

public:

 Student() {
 total = 0;
 average = 0;
 grade = 'F';
studentCount++;

}



void input() {

cout << "Enter Roll Number: ";

cin >> rollNumber;

cin.ignore();

cout << "Enter Name: ";

getline(cin, name);

total = 0;

for (int i = 0; i < 3; i++) {

cout << "Enter marks for subject " << (i + 1) << ": ";

cin >> marks[i];

total += marks[i];

}

average = total / 3;

calculateGrade();

}
```

```
void calculateTotalAndAverage() {

total = 0;

for (int i = 0; i < 3; i++) {

total += marks[i];

}

average = total / 3;

calculateGrade();

}


void display() const {

cout << "Roll Number: " << rollNumber << "\n";

cout << "Name: " << name << "\n";

for (int i = 0; i < 3; i++) {

cout << "Marks in subject " << (i + 1) << ": " << marks[i] << "\n";  }

cout << "Total: " << total << "\n";

cout << "Average: " << average << "\n";
cout << "Grade: " << grade << "\n";

}


float getAverage() const {

return average;

}


int getRollNumber() const {

return rollNumber;

}


float getTotal() const {

return total;

}


static int getStudentCount() {

return studentCount;

}
```

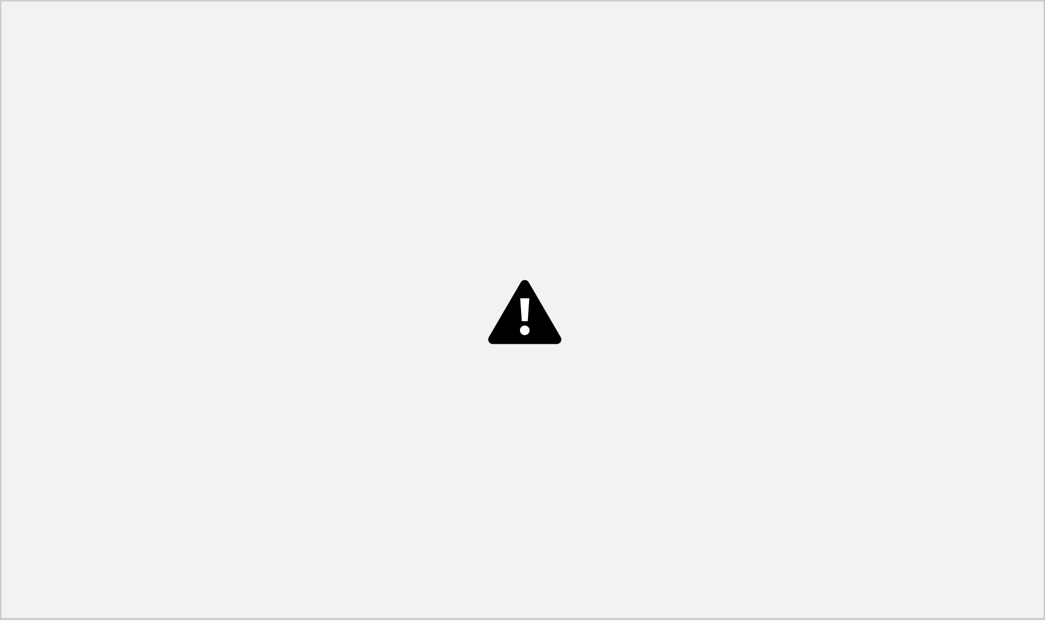
```
};


int Student::studentCount = 0;


int findHighestAverage(const vector<Student>& students) {
 int idx = 0;
 float highestAvg = students[0].getAverage();
 for (int i = 1; i < (int)students.size(); i++) {
 if (students[i].getAverage() > highestAvg) {
 highestAvg = students[i].getAverage();
 idx = i;
 }
 }
 return idx;
}


void sortByTotalMarks(vector<Student>& students) {
 for (int i = 0; i < (int)students.size() - 1; i++) {
```

```
 for (int j = 0; j < (int)students.size() - i - 1; j++) {

 if (students[j].getTotal() < students[j + 1].getTotal()) {

 swap(students[j], students[j + 1]);

 }

 }

 }

}


int searchByRollNumber(const vector<Student>& students, int rollNo) {

for (int i = 0; i < (int)students.size(); i++) {

 if (students[i].getRollNumber() == rollNo) {

 return i;

 }

 }

 return -1;

}


int main() {
```

```
int n;

cout << "Enter number of students: ";

cin >> n;


vector<Student> students(n);


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

cout << "\nEnter details for student " << (i + 1) << ":\n";

students[i].input();

}


cout << "\nAll Student Details:\n";

for (const auto& student : students) {

student.display();

cout << endl;

}


int highestIdx = findHighestAverage(students);
```

```
cout << "\nStudent with highest average:\n";

students[highestIdx].display();



sortByTotalMarks(students);



cout << "\nStudents sorted by total marks (descending):\n";

for (const auto& student : students) {

student.display();

cout << endl;

}



int rollToSearch;

cout << "Enter roll number to search: ";

cin >> rollToSearch;

int foundIdx = searchByRollNumber(students, rollToSearch);

if (foundIdx != -1) {

cout << "\nStudent found:\n";

students[foundIdx].display();
} else {

cout << "Student with roll number " << rollToSearch << " not found.\n";  }



cout << "\nTotal Students Created: " << Student::getStudentCount() << "\n";



return 0;

}
```

Output:



Conclusion:

Date:22-08-25 Signature of faculty in-charge

<u>Post Lab Descriptive Questions:</u>

What is encapsulation, and how is it implemented in your program? What is the difference between a constructor and a regular member function? Why are data members declared `private` in a class?

Output:

1. Encapsulation is one of the fundamental principles of Object-Oriented Programming (OOP). It means wrapping data (variables) and methods (functions) that operate on the data into a single unit (class) and restricting direct access to some of the object's components.

The main goal is to hide the internal state of an object and only allow manipulation through well-defined interfaces (public functions).

This protects the integrity of the data by preventing external code from directly changing the object's internal state in unexpected ways.

In the program:

- The student's data members like rollNumber, name, marks, total, average, and grade are declared private.

- They cannot be accessed or modified directly outside the class.

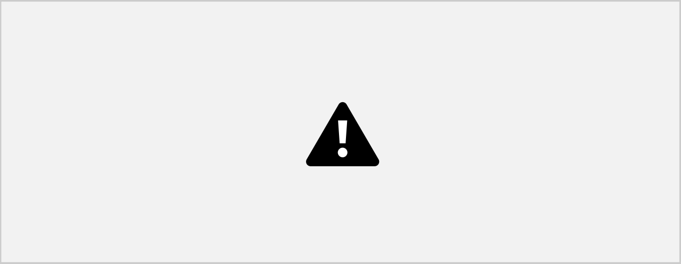
- Instead, we provide public member functions like `input()`, `display()`, and `calculateTotalAndAverage()` which control how the data is accessed or modified.

This ensures that the data is accessed in a controlled and safe way, maintaining the object's integrity.

2. Constructor:

- Purpose: To initialize an object when it is created

- Name: Has the same name as the class and no return type (not even void)



- Called automatically: Called automatically when an object is created
- Number of calls: Called once per object creation
- Example: `Student()` initializes data members

Regular Member Function:

- Purpose: To perform operations on objects after creation
- Name: Can have any valid function name and must specify return type
- Called automatically: Must be called explicitly using the object ●

Number of calls: Can be called multiple times as needed

- Example: `input()`, `display()` etc. used for other operations

3. Data members are declared private to:

- Protect data integrity: Prevent direct access/modification by outside code, which could lead to invalid or inconsistent states.

- Enforce encapsulation: Only allow controlled access via public member functions that can validate data or maintain class invariants.

- Enhance security: Hide implementation details, exposing only what is necessary.

- Enable flexibility: You can change the internal representation later without affecting code that uses the class, as external code interacts through a fixed interface.