CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Engineering

Subject Name: Object Oriented Programming with C++ Semester: II
Subject Code: CE144
Academic year: 2023-24

No.	Aim of the Practical
1.	

PROGRAM CODE :

Req1	Convert employee experience to float value
Inputs	Take employee experience (no of year and no of months)
Output	Enter employee experience in years and months Years :11 Months :8 Employee experience is :11.67 Enter employee experience in years and months Years :12 Months :6 Employee experience is :12.50

```
#include <iostream>

using namespace std;

class EmployeeExperience
{ private:
   float totalExperience;

public:
   // Default constructor
```

```
    EmployeeExperience() : totalExperience(0.0f) {}

    // Parameterized constructor
    EmployeeExperience(int years, int months) {
        totalExperience = years + months / 12.0f;
    }

    void display() {
        cout << "Employee experience is " << totalExperience << " years" << endl;
    }
};

int main() {
    int years, months;
    cout << "Enter employee experience in years: ";
    cin >> years;
    cout << "Enter employee experience in months: ";
    cin >> months;

    EmployeeExperience empExp(years, months);
    empExp.display();

    return 0;
}
```

OUTPUT:

```
Enter employee experience in years: 5
Enter employee experience in months: 2
Employee experience is: 5.16667

Process returned 0 (0x0)   execution time : 6.196 s
Press any key to continue.
```

CONCLUSION:

In this C++ program, I've demonstrated how to utilize a class to encapsulate the functionality for calculating an employee's experience in terms of years and months. The `EmployeeExperience` class contains private member variables for storing the years and months of experience, a constructor to initialize these variables, and a method `calculateExperience()` to compute the total experience in float value by converting months into a fractional part of a year.

	By encapsulating this functionality within a class, the code becomes more organized, modular, and easier to understand. It adheres to the principles of object-oriented programming (OOP) by encapsulating related data and functionality within a single entity.