ALLIANCE
UNIVERSITY

Alliance School of Advance Computing

Object Oriented Programming
Mini Project

System for managing different types of employees in an organization

Alliance University

Central Campus, Chikkahadage Cross Chandapura-Anekal, Main Road, Bengaluru, Karnataka 562106

Object Oriented Programming

3CS 1060

Mini Project

Submitted by:

Name of the students:

Name 1: KRUTHIKA L	2023BCSE07AED125
Name 2: TEJASWI KAISETTY	2023BCSE07AED141
Name 3: THANU SHREE M	2023BCSE07AED168
Name 4: SHREYA KUMARI	2023BCSE07AED202

Under the Guidance of

Dr/Prof. R.Rajasekar

ALLIANCE UNIVERSITY

Private University established in Karnataka State by Act No.34 of year 2010
Recognized by the University Grants Commission (UGC), New Delhi

Alliance School of Advance Computing

ACKNOWLEDGEMENT

I would like to express our sincere gratitude to Dr. R. Rajasekar for their invaluable guidance, encouragement, and support throughout the development of this project. Their expertise in C++ programming has been instrumental in helping me understand complex concepts such as templates, data structures and more. I am also deeply thankful to Alliance University and its Computer Science Department for providing us with an excellent learning environment and the resources necessary to enhance our knowledge and skills. Lastly, I would like to express our deepest gratitude to God for providing us with the strength, perseverance, and guidance to complete this project. Without his blessings, this project would not have been possible. Our teams comprise 4 members. Thank you to everyone who supported us in completing this project successfully.

Name 1: KRUTHIKA L 2023BCSE07AED125

Name 2: TEJASWI KAISETTY 2023BCSE07AED141

Name 3: THANU SHREE M 2023BCSE07AED168

Name 4: SHREYA KUMARI 2023BCSE07AED202

Signature of Mentor:

Dr. R. Rajasekar

Associate Professor in Computer Science & Engineering

INDEX

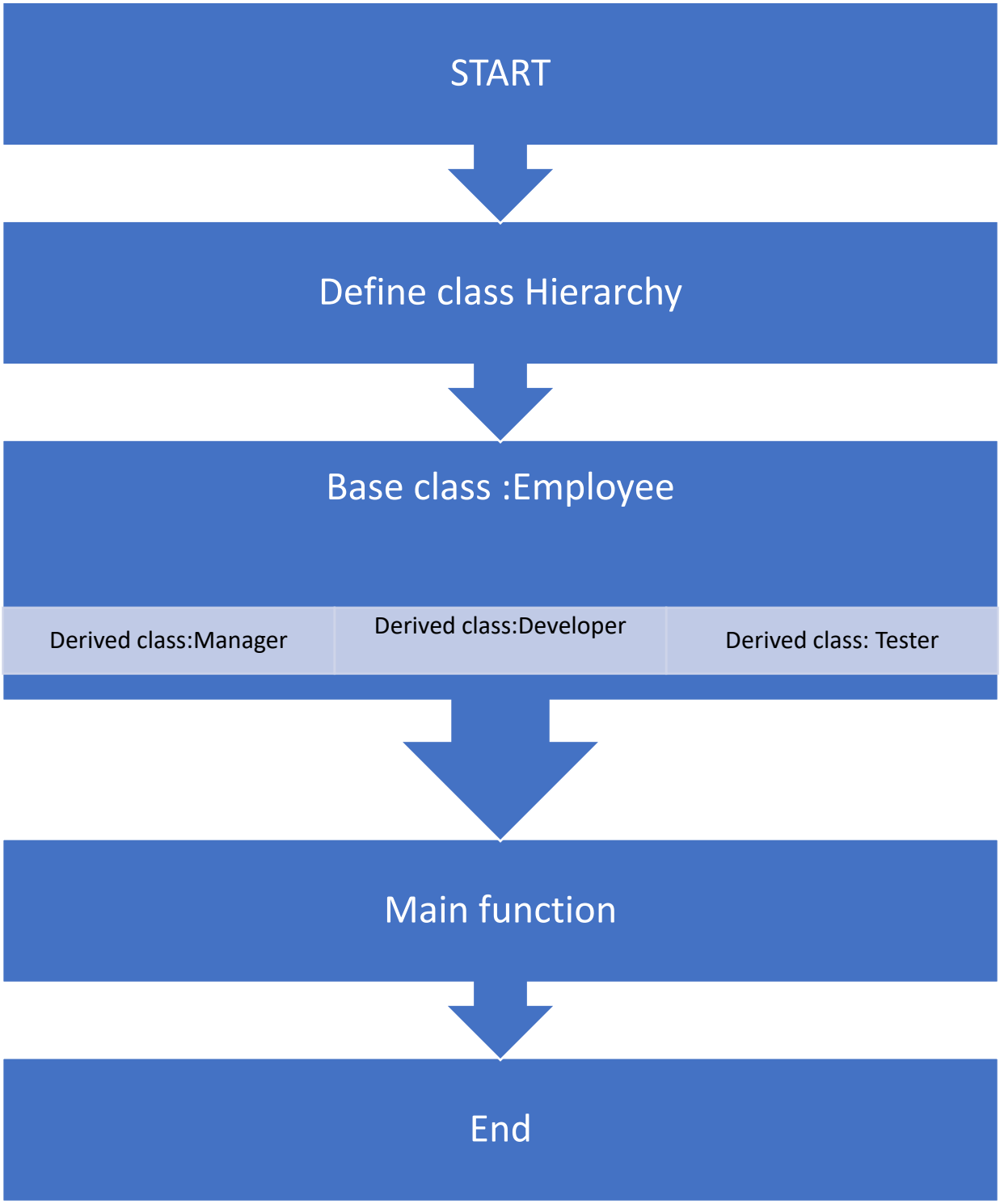
Problem statement

Create a system for managing different types of employees in an organization using inheritance and polymorphism in C++. Define a base class Employee with common attributes like name, ID, and salary. Derive classes Manager, Developer, and Tester from Employee, each with specific attributes and methods. Implement polymorphism by defining a virtual function calculate Salary () in the base class and overriding it in each derived class to compute salaries based on specific rules (e.g., including bonuses for managers, or overtime pay for developers).

Algorithm

1. Start
2. Describe the Employee base class:
   -Explain attributes such as name, ID, and pay.
   -To initialise the attributes, create a constructor.
   -Declare calculateSalary() as a pure virtual function that derived classes can override.
   -To print the employee details, define a function called display().
3. Define the Manager derived class:
   -Descended from the employee base class.
   -Incorporate an extra attribute bonus.
   -To return salary plus bonus, use the calculateSalary() function.
4. Define the Developer derived class:
   -Descended from the employee base class.
   -Incorporate overtimePay as an extra attribute.
   -To return salary plus overtime pay, use the calculateSalary() function.
5. Describe the Tester derived class:
   -Descended from the employee base class.
   -Incorporate an extra attribute called testingBonus.
   To return salary + testingBonus, use the calculateSalary() function.
6. In the main() function:
   -Create objects of Manager, Developer, and Tester by initializing their attributes (name, ID, salary, bonus/overtime/testing bonus).
   -Call the display() method for each object, which will also call the overridden calculateSalary() function for each employee type.
7. End.

FLOWCHART


START
Define class Hierarchy
Base class :Employee
Derived class:Manager
Derived class:Developer
Derived class: Tester
Main function
End

PROCEDURE

Step-by-Step Procedure:

Step 1: Define the Base Class (Employee)

The base class will define common attributes that are shared by all employees, such as:

- name: The name of the employee.
- ID: The unique identification number for the employee.
- salary: The base salary of the employee.

The base class will also define a virtual function calculateSalary() that will be overridden by derived classes to compute the salary according to specific rules.

Attributes:

- name (string): The employee's name.
- ID (int): The unique ID of the employee.
- salary (double): The base salary of the employee.

Methods:

- Constructor to initialize name, ID, and salary.
- Virtual method calculateSalary() that calculates the salary (to be customized in derived classes).

Step 2: Define Derived Classes (Manager, Developer, Tester)

Each derived class will inherit from Employee and will have additional attributes specific to that employee type.

1. **Manager**:
    a. Add an attribute bonusPercentage to represent the bonus percentage.
    b. Override the calculateSalary() method to include a bonus based on the salary.
2. **Developer**:
    a. Add attributes overtimeHours and overtimeRate.
    b. Override the calculateSalary() method to include overtime pay based on the hours worked and overtime rate.
3. **Tester**:
    a. Add an attribute performanceBonus to represent the bonus for performance.
    b. Override the calculateSalary() method to include a fixed performance bonus.

Step 3: Use Polymorphism for Salary Calculation

In the main program, create instances of the derived classes (Manager, Developer, Tester) but store them as pointers to the base class (Employee). Use polymorphism to call the calculateSalary() method dynamically at runtime, which will invoke the correct version of the method depending on the type of the object.

Step 4: Print Employee Details

For each employee, print the details such as their name, ID, and the calculated salary. This can be done using a helper function that takes an Employee reference or pointer.

Step 5: Testing and Conclusion

Test the system with different employee objects and ensure that the salary calculations are correct for each type of employee. The program should work dynamically for any number of employee types derived from Employee.

SOURCE CODE

```cpp
#include <iostream>
#include <string>
#include <vector>

// Base class
class Employee {
protected:
    std::string name; // Employee name
    int id;           // Employee ID
    double baseSalary; // Base salary

public:
    Employee(const std::string& name, int id, double baseSalary)
        : name(name), id(id), baseSalary(baseSalary) {}

    virtual double calculateSalary() const = 0; // Pure virtual function

    virtual void displayInfo() const {
        std::cout << "Name: " << name << "\nID: " << id << "\nBase Salary: " << baseSalary << std::endl;
    }

    virtual ~Employee() {} // Virtual destructor
};
```

```cpp

// Derived class for Manager
class Manager : public Employee {
private:
    double bonus; // Bonus for the manager

public:
    Manager(const std::string& name, int id, double baseSalary, double bonus)
        : Employee(name, id, baseSalary), bonus(bonus) {}

    double calculateSalary() const override {
        return baseSalary + bonus; // Total salary includes base salary and bonus
    }

    void displayInfo() const override {
        Employee::displayInfo();
        std::cout << "Bonus: " << bonus << std::endl;
    }
};

// Derived class for Developer
class Developer : public Employee {
```

```cpp
class Developer : public Employee {
private:
    int overtimeHours; // Number of overtime hours worked
    double overtimeRate; // Pay rate for overtime

public:
    Developer(const std::string& name, int id, double baseSalary, int overtimeHours, double overtimeRate)
        : Employee(name, id, baseSalary), overtimeHours(overtimeHours), overtimeRate(overtimeRate) {}

    double calculateSalary() const override {
        return baseSalary + (overtimeHours * overtimeRate); // Total salary includes base salary and overtime pay
    }

    void displayInfo() const override {
        Employee::displayInfo();
        std::cout << "Overtime Hours: " << overtimeHours << "\nOvertime Rate: " << overtimeRate << std::endl;
    }
};

// Derived class for Tester
class Tester : public Employee {
private:
    double bonus; // Bonus for the tester
```

```cpp
private:
    double bonus; // Bonus for the tester

public:
    Tester(const std::string& name, int id, double baseSalary, double bonus)
        : Employee(name, id, baseSalary), bonus(bonus) {}

    double calculateSalary() const override {
        return baseSalary + bonus; // Total salary includes base salary and bonus
    }

    void displayInfo() const override {
        Employee::displayInfo();
        std::cout << "Bonus: " << bonus << std::endl;
    }
};

int main() {
    // Creating a vector to store employees
    std::vector<Employee*> employees;

    // Adding different types of employees to the vector
    employees.push_back(new Manager("Alice", 101, 75000, 10000));
    employees.push_back(new Developer("Bob", 102, 60000, 15, 50));
```

```
    // Creating a vector to store employees
    std::vector<Employee*> employees;

    // Adding different types of employees to the vector
    employees.push_back(new Manager("Alice", 101, 75000, 10000));
    employees.push_back(new Developer("Bob", 102, 60000, 15, 50));
    employees.push_back(new Tester("Charlie", 103, 50000, 5000));

    // Displaying information and calculating salaries for each employee
    for (const auto& emp : employees) {
        emp->displayInfo();
        std::cout << "Total Salary: " << emp->calculateSalary() << std::endl;
        std::cout << "-------------------------------" << std::endl;
    }

    // Clean up memory
    for (const auto& emp : employees) {
        delete emp;
    }

    return 0;
}
```

OUTPUT

```
Name: Alice
ID: 101
Base Salary: 75000
Bonus: 10000
Total Salary: 85000
---------------------------------
Name: Bob
ID: 102
Base Salary: 60000
Overtime Hours: 15
Overtime Rate: 50
Total Salary: 60750
---------------------------------
Name: Charlie
ID: 103
Base Salary: 50000
Bonus: 5000
Total Salary: 55000
---------------------------------
```

CONCLUSION

This C++ program effectively demonstrates the use of **inheritance** and **polymorphism** to manage different types of employees in an organization.

1. **Base Class (Employee)**: Defines the common attributes and a virtual method calculateSalary(), providing a general interface for all employee types.
2. **Derived Classes (Manager, Developer, Tester)**: Each of these classes inherits from Employee and overrides the calculateSalary() method to implement salary calculations according to their specific rules (e.g., bonuses for managers, overtime pay for developers, performance bonuses for testers).
3. **Polymorphism**: The program uses polymorphism to dynamically call the correct calculateSalary() method for each employee type, even though the objects are stored as pointers to the base class (Employee). This allows for flexibility and extensibility in managing various employee types.
4. **Extensibility**: New employee types can easily be added by deriving new classes from Employee and implementing the salary calculation logic. This structure makes it easy to maintain and expand the system.

This approach can be further expanded with more features like handling employee promotions, benefits, and other attributes, demonstrating the power of object-oriented design in real-world systems.