(5/10)

National University of Computer and Emerging Sciences (Lahore)			
Course:	OOP	Course code:	CS217
Section:	BSCS-2B	Semester:	Spring 2024
Duration:	40 minutes	TotalMarks:	10
Date:	6/5/24	ID:	A
Name:	Haleemah	Roll no:	230554

Question 1:

NOTE: Read the entire question first before attempting.

A construction company requires a paycheque management system. It employs different types of employees who are all paid differently as described below. All types of employees must have a function that calculates their salaries but an employee can only be one of three types; manager, engineer or salesperson. All employees "must" belong to at least one of these categories. An unclassified employee "cannot be paid a salary".

1. All `Employees` have the following attributes common:
   - `name` (string): The name of the employee.
   - `id` (int): The unique ID of the employee.
   - `baseSalary` (double):
2. Implement three derived classes: `Manager`, `Engineer`, and `Salesperson`, each inheriting from the `Employee` class with unique attributes:
   - For **`Manager`**: `department` (string), `bonus` (double), `calculateSalary()` method. They are paid a bonus in addition to their base salary.
   - For **`Engineer`**: `rate` (double), `numProjects` (int), `calculateSalary()` method. They are paid the product of their rate and no. of projects in addition to their base salary.
   - For **`Salesperson`**: `salesAchieved` (double), `commissionRate` (double), `calculateSalary()` method. They are paid a commission on each sale made in addition to base salary.
3. Implement default and parameterised constructors, destructors and a `calculateSalary()` method in each derived class to calculate the salary of the respective employee type based on the provided attributes.
4. Give output of the main given on the next page.

```
int main() {

    // Create employee objects
    vector<Employee*> employees;

    employees.push_back(new Manager("Razan Usman", 100, 1000, "CS", 2000.0));
    employees.push_back(new Engineer("Armaghan Atiq", 420, 1000, 10.0, 5));
    employees.push_back(new Salesperson("Abdullah Ijaz", 666, 1000, 10, 10));

    // Calculate and display salaries
    for (Employee* e : employees) {
        cout << "Name: " << e->name << endl;
        cout << "ID: " << e->id << endl;
        cout << "Base Salary: $" << e->baseSalary << endl;
        cout << "Total Salary: $" << e->calculateSalary() <<endl;
        cout << endl;
    }

    // Free memory
    for (Employee* e : employees) {
        delete employee;
    }
    return 0;
}
```

Output:

```
Name: Razan Usman
ID: 100
Base Salary: $ 1000
Total Salary: $ 3000

Name: Armaghan Atiq
ID: 420
Base Salary: $ 1000
Total Salary: $ ~~50000~~ 1050

Name: Abdullah Ijaz
ID: 666
Base Salary: $ 1000
Total Salary: $ 2000
```

```
    }
~Salesperson()
  { cout << "Destructor called for Salesperson"}
};
```

Haleemah Zaheer
OOP QUIZ

```
class Employees
{
  public:
    string name;
    int ID;
    double baseSalary;
    Employees()=
    { string name="n";
      int ID:0;
      double basesalary=1000;
    }
    Employees ( string ~~name~~, int ~~ID~~ Id, double basesalary)
    { ~~name=n;~~ n=name;
      ~~ID=Id;~~ Id=ID;
      basesalary=baseSalary;
    }
    ~Employees()= virtual -2
    {
      cout << " Destructor for Employees() "<<endl;
    }
};
```

CalcSalary in parent? -2

```
class Manager: public Employees
{
  private:
    string department;
    double bonus;
  public:
    virtual double calculateSalary();
    virtual ~Calculate Salary();
    Manager() : call parent constructor. here
    { ID=0 ; name="n";
      department = "None";
      bonus = 0.00;
    }
    Manager (int b, string dpt,
             int id, string n
    {
        b= bonus;        id= ID;
       dpt= department;   n=name;
    }
    ~Manager()
    {
      cout << "Destructor called for Manager";
    }
  double Calculate Salary()
  {
    double Salary;
    Salary = baseSalary + bonus;
    return Salary;
  }
};
```

```
class Engineer : public Employees
{
  private:
    double rate;
    int numProjects;
  public:
    double CalculateSalary();
    ~CalculateSalary();
    Engineer()
    {
      ID = 0;
      name = "n";
      depa rate = 0.00;
      numProjects = 0;
    }
    ~Engineer()
    {
      cout << "Destructor called for Engineer()";
    }
    Engineer(int b, int id,
      string n, double rate,
      double numProjects)
    { b = baseSalary;
      id = ID;
      n = name;
      rate = this.rate;
      numprojects = numProjects;
    }
double Engineers :: CalculateSalary()
{
  double Salary;
  Salary = baseSalary + (rate * numProjects);
  return Salary;
}
};
```

should call Employee Const. here

```
class Salesperson: public Employees
{
  private:
      double  salesAchieved;
      double  commissionRate;
  public:
      double  calculateSalary();
      ~calculateSalary();
   Salesperson()
   {
     ID=0;
     name="n";
     salesAchieved = 0.00;
     commissionRate = 0.00;
   }
   Salesperson (int id, string n, double b,
       double sales, double rate)
   {
      ID=id;   id=ID;
          n= name;
           b= baseSalary;
          sales = salesAchieved;
           rate= commissionRate;
    }
 double  calculateSalary()
   { double Salary;
     Salary = baseSalary + (salesAchieved *
                              commissionRate);
     return Salary;
   }
  ~Salesperson()
   { cout << "Destructor called for Salesperson"}
};
```