Assignment - 4

C++

Name: Kamithkar Vinod
Course: PG DAC AUGUST 2025
PRN: 250850320040
Form No: 250500480
Date: 09-10-2025

Problem 1: Smart Inventory System with Dynamic Memory and Inheritance

Task: Design a program to manage an inventory system for a store. Each item in the store belongs to a specific category (like Electronics or Groceries), but the data must be stored and managed without using virtual functions. You must handle object relationships, memory allocation, and cleanup properly.

Code:

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

using namespace std;

// ItemType helps us manage pointer-to-object relationships
   without virtual functions.
enum class ItemType {
    BASE,
    ELECTRONICS,
    GROCERIES
};

// ----------------------
// Base Class: Item
// ----------------------
class Item {
private:
    string name;
    int id;
```

```
22      float price;
23
24  protected:
25      int quantity;
26      ItemType type; // store the actual runtime type (set by
           derived classes)
27
28  public:
29      // Parameterized constructor
30      Item(const string& name_, int id_, float price_, int
           quantity_)
31          : name(name_), id(id_), price(price_), quantity(quantity_
              ), type(ItemType::BASE) {
32      }
33
34      // Display base item details
35      void display() const {
36          cout << left << setw(15) << name
37               << setw(8) << id
38               << setw(10) << fixed << setprecision(2) << price
39               << setw(10) << quantity;
40      }
41
42      // Return total value: price * quantity
43      float getTotalValue() const {
44          return price * quantity;
45      }
46
47      // Accessors
48      string getName() const { return name; }
49      int getId() const { return id; }
50      float getPrice() const { return price; }
51      ItemType getType() const { return type; }
52
53      // Destructor (non-virtual by requirement)
54      ~Item() {
55          cout << "Destroying Item (id=" << id << ", name=\"" <<
               name << "\")\n";
56      }
57  };
58
59  // ----------------------
60  // Derived Class: Electronics
61  // ----------------------
62  class Electronics : public Item {
63  private:
64      int warrantyMonths; // additional member
65
66  public:
67      Electronics(const string& name_, int id_, float price_, int
           quantity_, int warrantyMonths_)
```

```
68          : Item(name_, id_, price_, quantity_), warrantyMonths(
               warrantyMonths_) {
69          // mark derived type
70          // Access protected member 'type' from base class
71          type = ItemType::ELECTRONICS;
72      }
73
74      void display() const {
75          // Call base display for common fields
76          Item::display();
77          cout << setw(15) << warrantyMonths << endl;
78      }
79
80      int getWarranty() const { return warrantyMonths; }
81
82      ~Electronics() {
83          cout << "Destroying Electronics specific resources (id="
               << getId() << ")\n";
84      }
85  };
86
87  // ----------------------
88  // Derived Class: Groceries
89  // ----------------------
90  class Groceries : public Item {
91  private:
92      string expiryDate; // example: "2025-12-31"
93
94  public:
95      Groceries(const string& name_, int id_, float price_, int
           quantity_, const string& expiryDate_)
96          : Item(name_, id_, price_, quantity_), expiryDate(
               expiryDate_) {
97          type = ItemType::GROCERIES;
98      }
99
100     void display() const {
101         Item::display();
102         cout << setw(15) << expiryDate << endl;
103     }
104
105     string getExpiry() const { return expiryDate; }
106
107     ~Groceries() {
108         cout << "Destroying Groceries specific resources (id=" <<
                getId() << ")\n";
109     }
110 };
111
112 // ----------------------
113 // Helper: print header
```

```
// ----------------------
void printInventoryHeader() {
    cout << left << setw(15) << "Name" << setw(8) << "ID" << setw(10) << "Price" << setw(10) << "Qty";
    cout << setw(15) << "Extra" << endl;
    cout << string(58, '-') << endl;
}

// ----------------------
// Main demo
// ----------------------
int main() {
    // We'll keep pointers to base class but store runtime type in each object.
    vector<Item*> items;

    // Dynamically allocate different derived objects
    items.push_back(new Electronics("SmartPhone", 101, 19999.00f, 5, 24));    // warranty 24 months
    items.push_back(new Groceries("Rice", 201, 59.50f, 50, "2026-01-15"));    // expiry
    items.push_back(new Electronics("Laptop", 102, 55999.99f, 2, 12));
    items.push_back(new Groceries("Milk", 202, 45.00f, 30, "2025-10-20"));

    // Display inventory
    cout << "\nCurrent Inventory:\n";
    printInventoryHeader();
    for (const Item* it : items) {
        if (it->getType() == ItemType::ELECTRONICS) {
            // safe because we know actual type via stored enum
            const Electronics* e = static_cast<const Electronics*>(it);
            e->display();
        } else if (it->getType() == ItemType::GROCERIES) {
            const Groceries* g = static_cast<const Groceries*>(it);
            g->display();
        } else {
            // base-only item
            it->display();
            cout << endl;
        }
    }

    // Compute and print total inventory value
    float grandTotal = 0.0f;
    for (const Item* it : items) {
        grandTotal += it->getTotalValue();
    }
```

```
157     cout << "\nGrand Total Value of Inventory: INR " << fixed <<
            setprecision(2) << grandTotal << "\n";
158
159     // Proper cleanup: delete via the actual derived type pointer
            .
160     // IMPORTANT: since base destructor is NOT virtual, do NOT
            delete using 'delete items[i]' when
161     // items[i] is typed Item* and actually points to a derived
            object. We instead cast to derived
162     // pointer type and delete via that pointer.
163     cout << "\nCleaning up dynamically allocated objects:\n";
164     for (Item* it : items) {
165         ItemType t = it->getType();
166         if (t == ItemType::ELECTRONICS) {
167             Electronics* e = static_cast<Electronics*>(it);
168             delete e; // deletes derived then base destructor
169         } else if (t == ItemType::GROCERIES) {
170             Groceries* g = static_cast<Groceries*>(it);
171             delete g;
172         } else {
173             // If there were plain Item objects allocated
                    directly, we'd delete them like this:
174             delete it;
175         }
176     }
177
178     items.clear();
179     cout << "All objects cleaned up.\n";
180
181     return 0;
182 }
```

Output: 1—

```
V:\CDAC\CDAC_PG_DAC_PRACTICE\4_CPP\4_advanced_assignment>

Current Inventory:
Name           ID      Price     Qty       Extra
-------------------------------------------------------
SmartPhone     101     19999.00  5         24
Rice           201     59.50     50        2026-01-15
Laptop         102     55999.99  2         12
Milk           202     45.00     30        2025-10-20

Grand Total Value of Inventory: INR 216319.97

Cleaning up dynamically allocated objects:
Destroying Electronics specific resources (id=101)
Destroying Item (id=101, name="SmartPhone")
Destroying Groceries specific resources (id=201)
Destroying Item (id=201, name="Rice")
Destroying Electronics specific resources (id=102)
Destroying Item (id=102, name="Laptop")
Destroying Groceries specific resources (id=202)
Destroying Item (id=202, name="Milk")
All objects cleaned up.
```

Problem 2: Employee Payroll Management System (with Dynamic Bonus Calculation))

Task: Design a C++ program to manage employees of a company. Each employee has common details (name, ID, base salary), but different roles (e.g.,Manager, Developer) that determine their bonus. You must use classes, inheritance, encapsulation, constructors, destructors, and pointers to:

- Store and display employee information.
- Dynamically allocate memory for employees.
- Compute their total salary (base + bonus).
- Ensure proper cleanup of allocated memory.

Code:

```cpp
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

// ============================
// Base Class: Employee
// ============================
class Employee {
private:
    string name;
    int id;
    float baseSalary;

```

```
15 protected:
16     float bonus;
17
18 public:
19     // Parameterized Constructor
20     Employee(const string& name_, int id_, float baseSalary_)
21         : name(name_), id(id_), baseSalary(baseSalary_), bonus
              (0.0f) {}
22
23     // Virtual Function - Base version
24     virtual void calculateBonus() {
25         bonus = 0.0f;  // Base class: no bonus
26     }
27
28     // Virtual Display Function
29     virtual void display() const {
30         cout << left << setw(15) << name
31              << setw(8) << id
32              << setw(12) << fixed << setprecision(2) <<
                    baseSalary
33              << setw(12) << bonus
34              << setw(15) << "Employee"
35              << setw(12) << baseSalary + bonus
36              << endl;
37     }
38
39     // Virtual Destructor
40     virtual ~Employee() {
41         cout << "Destroying Employee object: " << id << " (" <<
               name << ")\n";
42     }
43
44 protected:
45     // Protected getters for derived class access
46     string getName() const { return name; }
47     int getId() const { return id; }
48     float getBaseSalary() const { return baseSalary; }
49 };
50
51 // ============================
52 // Derived Class: Manager
53 // ============================
54 class Manager : public Employee {
55 public:
56     Manager(const string& name_, int id_, float baseSalary_)
57         : Employee(name_, id_, baseSalary_) {}
58
59     void calculateBonus() override {
60         bonus = 0.40f * getBaseSalary();  // 40% of base salary
61     }
62
```

```
63      void display() const override {
64          cout << left << setw(15) << getName()
65               << setw(8) << getId()
66               << setw(12) << fixed << setprecision(2) <<
                    getBaseSalary()
67               << setw(12) << bonus
68               << setw(15) << "Manager"
69               << setw(12) << getBaseSalary() + bonus
70               << endl;
71      }
72
73      ~Manager() override {
74          cout << "Destroying Manager object: " << getId() << "\n";
75      }
76  };
77
78  // ============================
79  // Derived Class: Developer
80  // ============================
81  class Developer : public Employee {
82  public:
83      Developer(const string& name_, int id_, float baseSalary_)
84          : Employee(name_, id_, baseSalary_) {}
85
86      void calculateBonus() override {
87          bonus = 0.25f * getBaseSalary();  // 25% of base salary
88      }
89
90      void display() const override {
91          cout << left << setw(15) << getName()
92               << setw(8) << getId()
93               << setw(12) << fixed << setprecision(2) <<
                    getBaseSalary()
94               << setw(12) << bonus
95               << setw(15) << "Developer"
96               << setw(12) << getBaseSalary() + bonus
97               << endl;
98      }
99
100     ~Developer() override {
101         cout << "Destroying Developer object: " << getId() << "\n
                ";
102     }
103 };
104
105 // ============================
106 // Helper Function: Header
107 // ============================
108 void printHeader() {
109     cout << left
110          << setw(15) << "Name"
```

```
111             << setw(8)  << "ID"
112             << setw(12) << "BaseSalary"
113             << setw(12) << "Bonus"
114             << setw(15) << "Role"
115             << setw(12) << "TotalSalary"
116             << endl;
117     cout << string(74, '-') << endl;
118 }
119 
120 // ============================
121 // Main Function
122 // ============================
123 int main() {
124     int n;
125     cout << "Enter the number of employees: ";
126     cin >> n;
127 
128     Employee** employees = new Employee*[n]; // Array of base
            pointers
129 
130     for (int i = 0; i < n; ++i) {
131         cout << "\nEnter details for employee #" << i + 1 << ":\n
                ";
132 
133         string name;
134         int id, choice;
135         float salary;
136 
137         cout << "Name: ";
138         cin >> ws;
139         getline(cin, name);
140         cout << "ID: ";
141         cin >> id;
142         cout << "Base Salary: ";
143         cin >> salary;
144 
145         cout << "Select Role (1 - Manager, 2 - Developer): ";
146         cin >> choice;
147 
148         if (choice == 1)
149             employees[i] = new Manager(name, id, salary);
150         else
151             employees[i] = new Developer(name, id, salary);
152 
153         // Calculate bonus for each employee
154         employees[i]->calculateBonus();
155     }
156 
157     // Display all employee details
158     cout << "\n\nEmployee Payroll Details:\n";
159     printHeader();
```

```
    for (int i = 0; i < n; ++i) {
        employees[i]->display();
    }

    // Cleanup
    cout << "\nCleaning up memory:\n";
    for (int i = 0; i < n; ++i) {
        delete employees[i];
    }
    delete[] employees;

    cout << "All employee objects destroyed successfully.\n";
    return 0;
}
```

Output: —

```
V:\CDAC\CDAC_PG_DAC_PRACTICE\4_CPP\4_advanced_assignment>employeePayrollManagement
Enter the number of employees: 2

Enter details for employee #1:
Name: Vinod
ID: 1947
Base Salary: 2500000
Select Role (1 - Manager, 2 - Developer): 1

Enter details for employee #2:
Name: Sony
ID: 1948
Base Salary: 2400000
Select Role (1 - Manager, 2 - Developer): 2


Employee Payroll Details:
Name           ID      BaseSalary  Bonus       Role           TotalSalary
--------------------------------------------------------------------------
Vinod          1947    2500000.00  1000000.00  Manager        3500000.00
Sony           1948    2400000.00  600000.00   Developer      3000000.00

Cleaning up memory:
Destroying Manager object: 1947
Destroying Employee object: 1947 (Vinod)
Destroying Developer object: 1948
Destroying Employee object: 1948 (Sony)
All employee objects destroyed successfully.
```

Problem 3: Menu-Driven Employee Management System using Classes, Objects, Inheritance, and Dynamic Memory in C++

Task: Design a Menu-Driven Employee Management System for a company that manages two types of employees: 1. FullTimeEmployee

2. PartTimeEmployee

You must: Use inheritance to derive these two classes from a base class Employee. Use encapsulation for data hiding (private/protected members). Create objects dynamically using pointers. Display and manage data using a menu-driven interface.

Code: —

```
#include <iostream>
```

```cpp
#include <string>
#include <vector>
#include <iomanip>
using namespace std;

// ======================================
// Base Class: Employee
// ======================================
class Employee {
private:
    string name;
    int empID;

protected:
    float salary;

public:
    // Parameterized constructor
    Employee(const string& name_, int empID_)
        : name(name_), empID(empID_), salary(0.0f) {}

    // Display basic info
    void displayBasic() const {
        cout << left << setw(15) << name
             << setw(10) << empID;
    }

    // Getter
    float getSalary() const { return salary; }

    int getID() const { return empID; }
    string getName() const { return name; }

    // Virtual destructor
    virtual ~Employee() {
        cout << "Destroying Employee: " << empID << " (" << name
           << ")\n";
    }

    // Virtual functions to override
    virtual void calculateSalary() = 0;
    virtual void displayDetails() const = 0;
};

// ======================================
// Derived Class: FullTimeEmployee
// ======================================
class FullTimeEmployee : public Employee {
private:
    float basicPay;
    float bonus;
```

```
52
53 public:
54     FullTimeEmployee(const string& name_, int empID_, float
          basicPay_, float bonus_)
55         : Employee(name_, empID_), basicPay(basicPay_), bonus(
              bonus_) {}
56
57     void calculateSalary() override {
58         salary = basicPay + bonus;
59     }
60
61     void displayDetails() const override {
62         displayBasic();
63         cout << setw(15) << "Full-Time"
64              << setw(12) << fixed << setprecision(2) << salary
65              << setw(12) << basicPay
66              << setw(12) << bonus
67              << endl;
68     }
69
70     ~FullTimeEmployee() override {
71         cout << "Cleaning up FullTimeEmployee object (ID=" <<
              getID() << ")\n";
72     }
73 };
74
75 // =====================================
76 // Derived Class: PartTimeEmployee
77 // =====================================
78 class PartTimeEmployee : public Employee {
79 private:
80     int hoursWorked;
81     float hourlyRate;
82
83 public:
84     PartTimeEmployee(const string& name_, int empID_, int
          hoursWorked_, float hourlyRate_)
85         : Employee(name_, empID_), hoursWorked(hoursWorked_),
              hourlyRate(hourlyRate_) {}
86
87     void calculateSalary() override {
88         salary = hoursWorked * hourlyRate;
89     }
90
91     void displayDetails() const override {
92         displayBasic();
93         cout << setw(15) << "Part-Time"
94              << setw(12) << fixed << setprecision(2) << salary
95              << setw(12) << hoursWorked
96              << setw(12) << hourlyRate
97              << endl;
```

```
98      }
99
100     ~PartTimeEmployee() override {
101         cout << "Cleaning up PartTimeEmployee object (ID=" <<
                getID() << ")\n";
102     }
103 };
104
105 // =====================================
106 // Helper Function: Display Header
107 // =====================================
108 void displayHeader() {
109     cout << left << setw(15) << "Name"
110          << setw(10) << "EmpID"
111          << setw(15) << "Type"
112          << setw(12) << "Salary"
113          << setw(12) << "Basic/Hrs"
114          << setw(12) << "Bonus/Rate"
115          << endl;
116     cout << string(76, '-') << endl;
117 }
118
119 // =====================================
120 // Search Function
121 // =====================================
122 Employee* searchEmployee(vector<Employee*>& employees, int id) {
123     for (auto* e : employees) {
124         if (e->getID() == id)
125             return e;
126     }
127     return nullptr;
128 }
129
130 // =====================================
131 // Delete Function
132 // =====================================
133 void deleteEmployee(vector<Employee*>& employees, int id) {
134     for (auto it = employees.begin(); it != employees.end(); ++it
           ) {
135         if ((*it)->getID() == id) {
136             delete *it; // Free memory
137             employees.erase(it);
138             cout << "Employee with ID " << id << " deleted
                    successfully.\n";
139             return;
140         }
141     }
142     cout << "Employee with ID " << id << " not found.\n";
143 }
144
145 // =====================================
```

```
146 // Main Function
147 // ======================================
148 int main() {
149     vector<Employee*> employees;
150     int choice;
151 
152     do {
153         cout << "\n=========== Employee Management System
                ===========\n";
154         cout << "1. Add Full-Time Employee\n";
155         cout << "2. Add Part-Time Employee\n";
156         cout << "3. Display All Employees\n";
157         cout << "4. Search Employee by ID\n";
158         cout << "5. Delete Employee by ID\n";
159         cout << "6. Exit\n";
160         cout << "Enter your choice: ";
161         cin >> choice;
162 
163         switch (choice) {
164         case 1: {
165             string name;
166             int id;
167             float basic, bonus;
168             cout << "Enter name: ";
169             cin >> ws;
170             getline(cin, name);
171             cout << "Enter ID: ";
172             cin >> id;
173             cout << "Enter Basic Pay: ";
174             cin >> basic;
175             cout << "Enter Bonus: ";
176             cin >> bonus;
177 
178             Employee* e = new FullTimeEmployee(name, id, basic,
                    bonus);
179             e->calculateSalary();
180             employees.push_back(e);
181             cout << "Full-Time Employee added successfully!\n";
182             break;
183         }
184 
185         case 2: {
186             string name;
187             int id, hours;
188             float rate;
189             cout << "Enter name: ";
190             cin >> ws;
191             getline(cin, name);
192             cout << "Enter ID: ";
193             cin >> id;
194             cout << "Enter Hours Worked: ";
```

```
            cin >> hours;
            cout << "Enter Hourly Rate: ";
            cin >> rate;

            Employee* e = new PartTimeEmployee(name, id, hours,
               rate);
            e->calculateSalary();
            employees.push_back(e);
            cout << "Part-Time Employee added successfully!\n";
            break;
        }

        case 3: {
            if (employees.empty()) {
                cout << "No employees to display.\n";
            } else {
                displayHeader();
                for (auto* e : employees)
                    e->displayDetails();
            }
            break;
        }

        case 4: {
            int id;
            cout << "Enter Employee ID to search: ";
            cin >> id;
            Employee* emp = searchEmployee(employees, id);
            if (emp) {
                displayHeader();
                emp->displayDetails();
            } else {
                cout << "Employee not found.\n";
            }
            break;
        }

        case 5: {
            int id;
            cout << "Enter Employee ID to delete: ";
            cin >> id;
            deleteEmployee(employees, id);
            break;
        }

        case 6: {
            cout << "Exiting program...\n";
            break;
        }

        default:
```

```
            cout << "Invalid choice. Please try again.\n";
        }

    } while (choice != 6);

    // Cleanup before exit
    cout << "\nCleaning up all remaining employee objects...\n";
    for (auto* e : employees)
        delete e;
    employees.clear();

    cout << "All memory released successfully.\n";
    return 0;
}
```

Output: —

```
V:\CDAC\CDAC_PG_DAC_PRACTICE\4_CPP\4_advanced_assignment>

========== Employee Management System ==========
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Search Employee by ID
5. Delete Employee by ID
6. Exit
Enter your choice: 1
Enter name: Vinod
Enter ID: 101
Enter Basic Pay: 250000
Enter Bonus: 25000
Full-Time Employee added successfully!

========== Employee Management System ==========
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Search Employee by ID
5. Delete Employee by ID
6. Exit
Enter your choice: 2
Enter name: Sony
Enter ID: 102
Enter Hours Worked: 43
Enter Hourly Rate: 24000
Part-Time Employee added successfully!
```

```
========== Employee Management System ==========
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Search Employee by ID
5. Delete Employee by ID
6. Exit
Enter your choice: 3
Name           EmpID     Type           Salary      Basic/Hrs   Bonus/Rate
-------------------------------------------------------------------------
Vinod          101       Full-Time      275000.00   250000.00   25000.00
Sony           102       Part-Time      1032000.00  43          24000.00

========== Employee Management System ==========
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Search Employee by ID
5. Delete Employee by ID
6. Exit
Enter your choice: 5
Enter Employee ID to delete: 101
Cleaning up FullTimeEmployee object (ID=101)
Destroying Employee: 101 (Vinod)
Employee with ID 101 deleted successfully.

========== Employee Management System ==========
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Search Employee by ID
5. Delete Employee by ID
6. Exit
Enter your choice: 6
Exiting program...

Cleaning up all remaining employee objects...
Cleaning up PartTimeEmployee object (ID=102)
Destroying Employee: 102 (Sony)
All memory released successfully.
```