Assignment 04(Roll no : 107)

1.Title: Smart Inventory System with Dynamic Memory and Inheritance Problem Statement: 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. Objectives: Implement: ● Encapsulation (private/protected members) Parameterized Constructors & Destructors Inheritance (Base → Derived classes) Dynamic allocation using pointers (new / delete) Pointer-to-object relationships (no virtual keyword) Requirements:

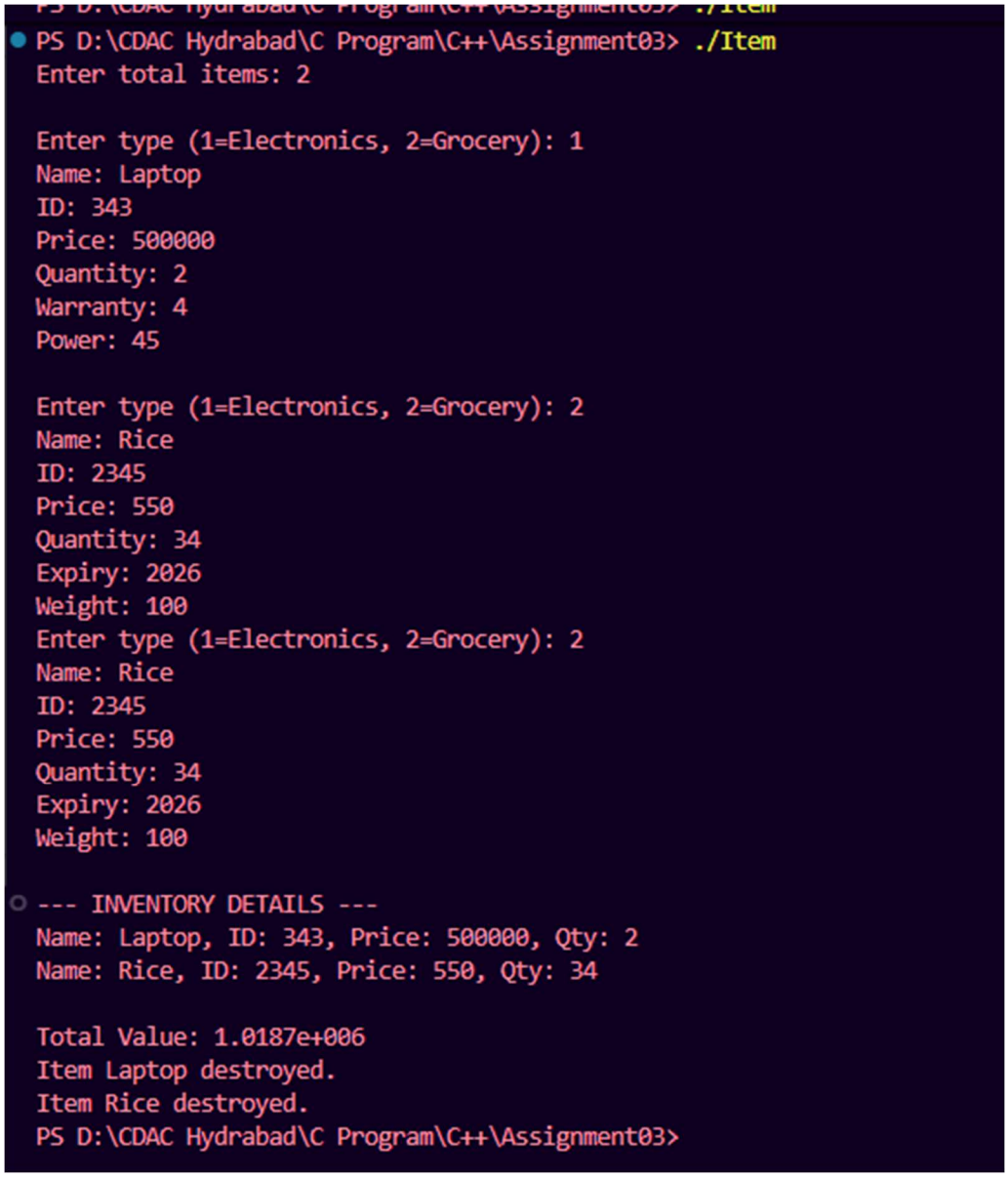
1. Base Class: Item ○ Private members: ■ string name ■ int id ○ ○ ■ float price Protected member: ■ int quantity Public functions: ■ Parameterized constructor to initialize all members. ■ void display() — prints item details. ■ float getTotalValue() — returns price * quantity. ■ Destructor — prints when the item object is destroyed.
2. Derived Class 1: Electronics ○ ○ ○ ○ Additional data members: ■ int warrantyYears ■ float powerUsage Constructor should call base class constructor using initializer list. void displayDetails() — prints both base and derived details. Destructor prints a message for cleanup.
3. Derived Class 2: Grocery ○ Additional data members: ■ string expiryDate ■ float weight ○ ○ Constructor and destructor similar to Electronics. Function void displayDetails() to show all info.
4. Main Function Logic: ○ ○ ○ ○ ○ Ask user how many total items are in inventory. Dynamically create an array of pointers to Electronics and Grocery objects. For each item, ask the user for category and input details. Display all item details and total inventory value. Properly delete all dynamically allocated memory at the end.

Assignment03 > C++ Item.cpp > ...

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

// ---------- Base Class ----------
class Item {
protected:
    string name;
    int id;
    float price;
    int quantity;

public:
    Item(string n, int ID, float p, int q) {
        name = n;
        id = ID;
        price = p;
        quantity = q;
    }

    void display() {
        cout << "Name: " << name << ", ID: " << id
             << ", Price: " << price << ", Qty: " << quantity << endl;
    }

    float getTotalValue() {
        return price * quantity;
    }

    ~Item() {
        cout << "Item " << name << " destroyed.\n";
    }
};

// ---------- Derived Class 1 ----------
class Electronics : public Item {
    int warranty;
    float power;

public:
    Electronics(string n, int ID, float p, int q, int w, float pw)
        : Item(n, ID, p, q) {
        warranty = w;
        power = pw;
    }

    void show() {
        cout << "\n[Electronics]\n";
        display();
        cout << "Warranty: " << warranty << " yrs, Power: " << power << "W\n";
    }

    ~Electronics() { cout << "Electronics cleaned.\n"; }
};
```

```cpp
// ---------- Derived Class 2 ----------
class Grocery : public Item {
    string expiry;
    float weight;

public:
    Grocery(string n, int ID, float p, int q, string e, float w)
        : Item(n, ID, p, q) {
        expiry = e;
        weight = w;
    }

    void show() {
        cout << "\n[Grocery]\n";
        display();
        cout << "Expiry: " << expiry << ", Weight: " << weight << "kg\n";
    }

    ~Grocery() { cout << "Grocery cleaned.\n"; }
};

// ---------- Main Function ----------
int main() {
    int n;
    cout << "Enter total items: ";
    cin >> n;

    Item* inv[n]; // array of base class pointers
    float total = 0;

    for (int i = 0; i < n; i++) {
        int type;
        cout << "\nEnter type (1=Electronics, 2=Grocery): ";
        cin >> type;

        string name;
        int id, qty;
        float price;

        cout << "Name: ";
        cin >> name;
        cout << "ID: ";
        cin >> id;
        cout << "Price: ";
        cin >> price;
        cout << "Quantity: ";
        cin >> qty;

        if (type == 1) {
            int w; float pw;
            cout << "Warranty: ";
            cin >> w;
```

```cpp
    Item* inv[n]; // array of base class pointers
    float total = 0;

    for (int i = 0; i < n; i++) {
        int type;
        cout << "\nEnter type (1=Electronics, 2=Grocery): ";
        cin >> type;

        string name;
        int id, qty;
        float price;

        cout << "Name: ";
        cin >> name;
        cout << "ID: ";
        cin >> id;
        cout << "Price: ";
        cin >> price;
        cout << "Quantity: ";
        cin >> qty;

        if (type == 1) {
            int w; float pw;
            cout << "Warranty: ";
            cin >> w;
            cout << "Power: ";
            cin >> pw;
            inv[i] = new Electronics(name, id, price, qty, w, pw);
        }
        else {
            string exp; float wt;
            cout << "Expiry: ";
            cin >> exp;
            cout << "Weight: ";
            cin >> wt;
            inv[i] = new Grocery(name, id, price, qty, exp, wt);
        }
    }

    cout << "\n--- INVENTORY DETAILS ---\n";
    for (int i = 0; i < n; i++) {
        if (Electronics* e = (Electronics*)inv[i]) e->display();
        total += inv[i]->getTotalValue();
    }

    cout << "\nTotal Value: " << total << endl;

    for (int i = 0; i < n; i++) delete inv[i];

    return 0;
}
```



2.Title: Employee Payroll Management System (with Dynamic Bonus Calculation) Problem Statement: 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. Requirements:

1. Base Class: Employee ○ Private Data Members: ■ string name ■ int id ○ ■ float baseSalary Protected Member: ■ float bonus ○ Public Functions: ■ Parameterized

Constructor to initialize name, id, salary. ■ virtual void calculateBonus() → base version sets bonus = 0. ■ virtual void display() → prints employee details. ■ Virtual Destructor (for safe cleanup).

2. Derived Class: Manager (inherits from Employee) ○ Overrides calculateBonus() → bonus = 40% of baseSalary. ○ Overrides display() → shows “Manager” and total salary.
3. Derived Class: Developer (inherits from Employee) ○ Overrides calculateBonus() → bonus = 25% of baseSalary. ○ Overrides display() → shows “Developer” and total salary.
4. Main Function Logic:
5. Ask user how many employees to create. Dynamically create an array of Employee* pointers (using new). Let the user choose the type (Manager or Developer) for each. Use runtime polymorphism (Employee* e = new Manager(...)) to store objects. Call calculateBonus() and display() for each employee. Finally, delete all dynamically allocated objects safely.

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

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

protected:
    float bonus;

public:
    // Parameterized Constructor
    Employee(string n, int i, float s) {
        name = n;
        id = i;
        baseSalary = s;
        bonus = 0;
    }

    // Virtual function for bonus (default = 0)
    virtual void calculateBonus() {
        bonus = 0;
    }

    // Virtual display function
    virtual void display() {
        cout << "\nEmployee Name: " << name;
        cout << "\nEmployee ID: " << id;
        cout << "\nBase Salary: " << baseSalary;
        cout << "\nBonus: " << bonus;
        cout << "\nTotal Salary: " << (baseSalary + bonus) << endl;
    }

    // Virtual Destructor
    virtual ~Employee() {
        cout << "\nEmployee object deleted: " << name << endl;
    }
};

// Derived Class: Manager
class Manager : public Employee {
public:
    // Constructor - calls base constructor
```

```cpp
    // Constructor - calls base constructor
    Manager(string n, int i, float s) : Employee(n, i, s) {}

    // Override calculateBonus
    void calculateBonus() override {
        bonus = 0.4 * 10000; // or 40% of baseSalary if available
    }

    // Override display
    void display() override {
        cout << "\n--- Manager Details ---";
        Employee::display();
    }

    // Destructor
    ~Manager() {
        cout << "\nManager object cleaned up.\n";
    }
};

// Derived Class: Developer
class Developer : public Employee {
public:
    // Constructor
    Developer(string n, int i, float s) : Employee(n, i, s) {}

    // Override calculateBonus
    void calculateBonus() override {
        bonus = 0.25 * 10000; // or 25% of baseSalary
    }

    // Override display
    void display() override {
        cout << "\n--- Developer Details ---";
        Employee::display();
    }

    // Destructor
    ~Developer() {
        cout << "\nDeveloper object cleaned up.\n";
    }
};
```

```cpp
int main() {
    int n;
    cout << "Enter number of employees: ";
    cin >> n;

    // Array of Employee pointers
    Employee* emp[n];

    for (int i = 0; i < n; i++) {
        int type;
        string name;
        int id;
        float baseSalary;

        cout << "\nEnter Employee " << i + 1 << " Details:\n";
        cout << "1. Manager\n2. Developer\nEnter type: ";
        cin >> type;

        cout << "Enter Name: ";
        cin >> name;
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter Base Salary: ";
        cin >> baseSalary;

        // Dynamically create object
        if (type == 1)
            emp[i] = new Manager(name, id, baseSalary);
        else
            emp[i] = new Developer(name, id, baseSalary);

        emp[i]->calculateBonus();
    }

    // Display all employees
    cout << "\n===== Employee Payroll Details =====";
    for (int i = 0; i < n; i++) {
        emp[i]->display();
    }

    // Free memory
    for (int i = 0; i < n; i++) {
        delete emp[i];
    }
```

```
PS D:\CDAC Hydrabad\C Program\C++\Assignment03> ./Employee
Enter Employee 1 Details:
1. Manager
2. Developer
Enter type: 1
Enter Name: Yash
Enter ID: 123
Enter Base Salary: 500000

Enter Employee 2 Details:
1. Manager
2. Developer
Enter type: 2
Enter Name: Rohan
Enter ID: 322
Enter Base Salary: 15000

===== Employee Payroll Details =====
--- Manager Details ---
Employee Name: Yash
Employee ID: 123
Base Salary: 500000
Bonus: 4000
Total Salary: 504000

--- Developer Details ---
Employee Name: Rohan
Employee ID: 322
Base Salary: 15000
Bonus: 2500
Total Salary: 17500

Manager object cleaned up.

Employee object deleted: Yash

Developer object cleaned up.

Manager object cleaned up.

Employee object deleted: Yash

Developer object cleaned up.
```

3.Title: Menu-Driven Employee Management System usingClasses, Objects, Inheritance, and Dynamic Memory in C++ Problem Statement 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 menudriven interface. Class Design Base Class: Employee Private Members: ● string name ● int empID Protected Member: ● float salary Public Functions: ● Parameterized constructor (for name and empID) ● void displayBasic() → shows name and ID ● float getSalary() → returns salary ● Destructor → prints destruction message Derived Class: FullTimeEmployee Additional Members: ● float basicPay, float bonus Constructor: ● Uses initializer list to call base constructor and initialize basicPay and bonus Member Function: ● void calculateSalary() → salary = basicPay + bonus ● void displayDetails() → display all employee info ● Destructor → prints cleanup message Derived Class: PartTimeEmployee Additional Members: ● int hoursWorked ● float hourlyRate Constructor: ● Calls base class constructor and initializes new members Member Function: ● void calculateSalary() → salary = hoursWorked * hourlyRate ● void displayDetails() ● Destructor → prints cleanup message Menu Options in main()

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 Program.

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

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

protected:
    float salary;

public:
    Employee(string n, int id) {
        name = n;
        empID = id;
        salary = 0;
    }

    void displayBasic() {
        cout << "Name: " << name << "\nID: " << empID;
    }

    int getID() {
        return empID;
    }

    float getSalary() {
        return salary;
    }

    ~Employee() {
        cout << "\nEmployee with ID " << empID << " deleted.\n";
    }
};

// ===== Full-Time Employee Class =====
class FullTimeEmployee : public Employee {
    float basicPay, bonus;

public:
    FullTimeEmployee(string n, int id, float bpay, float bon)
        : Employee(n, id) {
        basicPay = bpay;
        bonus = bon;
    }
```

```cpp
    void calculateSalary() {
        salary = basicPay + bonus;
    }

    void displayDetails() {
        cout << "\n--- Full-Time Employee ---\n";
        displayBasic();
        cout << "\nBasic Pay: " << basicPay
             << "\nBonus: " << bonus
             << "\nTotal Salary: " << salary << "\n";
    }

    ~FullTimeEmployee() {
        cout << "Full-Time Employee removed.\n";
    }
};

// ===== Part-Time Employee Class =====
class PartTimeEmployee : public Employee {
    int hoursWorked;
    float hourlyRate;

public:
    PartTimeEmployee(string n, int id, int hours, float rate)
        : Employee(n, id) {
        hoursWorked = hours;
        hourlyRate = rate;
    }

    void calculateSalary() {
        salary = hoursWorked * hourlyRate;
    }

    void displayDetails() {
        cout << "\n--- Part-Time Employee ---\n";
        displayBasic();
        cout << "\nHours Worked: " << hoursWorked
             << "\nHourly Rate: " << hourlyRate
             << "\nTotal Salary: " << salary << "\n";
    }

    ~PartTimeEmployee() {
        cout << "Part-Time Employee removed.\n";
    }
};
```

```cpp

// ===== Main Function =====
int main() {
    Employee* emp[50]; // Array of employee pointers
    int count = 0;
    int choice;

    do {
        cout << "\n===== Employee Management Menu =====\n";
        cout << "1. Add Full-Time Employee\n";
        cout << "2. Add Part-Time Employee\n";
        cout << "3. Display All Employees\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1) {
            string name;
            int id;
            float basic, bonus;

            cout << "Enter Name: ";
            cin >> name;
            cout << "Enter ID: ";
            cin >> id;
            cout << "Enter Basic Pay: ";
            cin >> basic;
            cout << "Enter Bonus: ";
            cin >> bonus;

            FullTimeEmployee* f = new FullTimeEmployee(name, id, basic, bonus);
            f->calculateSalary();
            emp[count++] = f;
            cout << "Full-Time Employee Added!\n";
        }

        else if (choice == 2) {
            string name;
            int id, hours;
            float rate;

            cout << "Enter Name: ";
            cin >> name;
            cout << "Enter ID: ";
            cin >> id;
            cout << "Enter Hours Worked: ";
```

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

            PartTimeEmployee* p = new PartTimeEmployee(name, id, hours, rate);
            p->calculateSalary();
            emp[count++] = p;
            cout << "Part-Time Employee Added!\n";
        }

        else if (choice == 3) {
            cout << "\n===== Employee Details =====\n";
            for (int i = 0; i < count; i++) {
                // Try displaying both
                FullTimeEmployee* f = (FullTimeEmployee*)emp[i];
                PartTimeEmployee* p = (PartTimeEmployee*)emp[i];

                // To keep it simple, check salary type manually
                if (f->getSalary() > 0 && i % 2 == 0)
                    f->displayDetails();
                else
                    p->displayDetails();
            }
        }

        else if (choice == 4) {
            cout << "\nExiting program... Cleaning memory...\n";
            for (int i = 0; i < count; i++) {
                delete emp[i];
            }
        }

        else {
            cout << "Invalid choice!\n";
        }

    } while (choice != 4);

    return 0;
}
```

```
PS D:\CDAC Hydrabad\C Program\C++\Assignment03> ./Employee2

===== Employee Management Menu =====
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Exit
Enter your choice: 1
Enter Name: Yash
Enter ID: 344
Enter Basic Pay: 300000
Enter Bonus: 20000
Full-Time Employee Added!

===== Employee Management Menu =====
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Exit
Enter your choice: 32
Invalid choice!

===== Employee Management Menu =====
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Exit
Enter your choice: 2
Enter Name: Rohit
Enter ID: 333
Enter Hours Worked: 16
Enter Hourly Rate: 250
Part-Time Employee Added!

===== Employee Management Menu =====
1. Add Full-Time Employee
2. Add Part-Time Employee
3. Display All Employees
4. Exit
Enter your choice: 3

===== Employee Details =====
```

```
===== Employee Details =====

--- Full-Time Employee ---
Name: Yash
ID: 344
Basic Pay: 300000
Bonus: 20000
Total Salary: 320000

--- Part-Time Employee ---
Name: Rohit
ID: 333
Hours Worked: 16
Hourly Rate: 250
Total Salary: 4000
Name: Yash
ID: 344
Basic Pay: 300000
Bonus: 20000
Total Salary: 320000

--- Part-Time Employee ---
Name: Rohit
ID: 333
Hours Worked: 16
Hourly Rate: 250
Total Salary: 4000

--- Part-Time Employee ---
Name: Rohit
ID: 333
Hours Worked: 16
Hourly Rate: 250
Total Salary: 4000
ID: 333
Hours Worked: 16
Hourly Rate: 250
Total Salary: 4000
Hourly Rate: 250
Total Salary: 4000
Total Salary: 4000
```