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.

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:**
    - 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.

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 **menu-driven 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