Assignment - 4(C++)

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.

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

// ================= BASE CLASS =================
class Item {
private:
    string name;
    int id;
    float price;

protected:
    int quantity;

public:
    // Simple parameterized constructor
    Item(string name_, int id_, float price_, int quantity_) {
        name = name_;
        id = id_;
        price = price_;
        quantity = quantity_;
    }

    // Display item details
    void display() const {
```

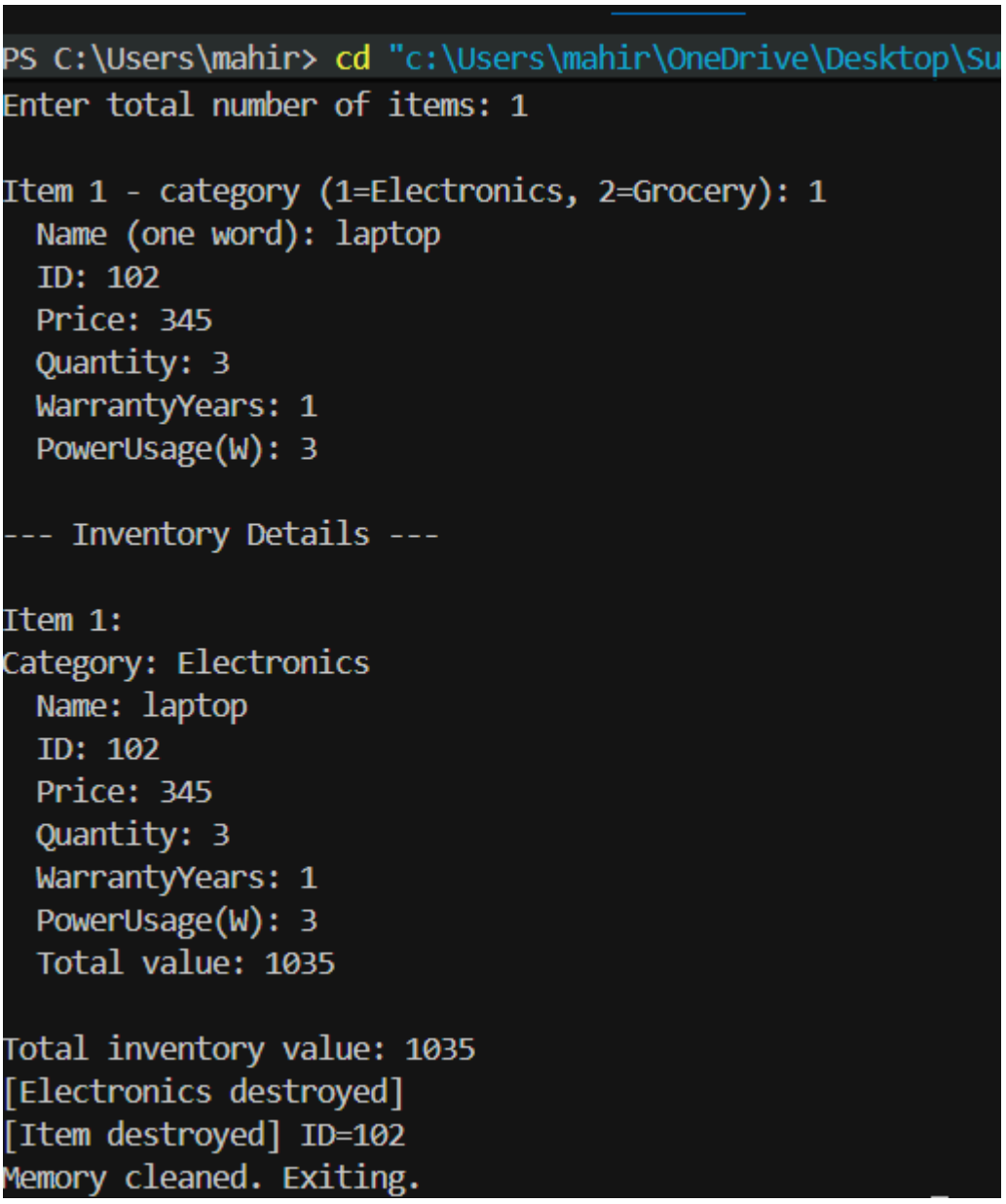
```
        cout << "  Name: " << name << endl;
        cout << "  ID: " << id << endl;
        cout << "  Price: " << price << endl;
        cout << "  Quantity: " << quantity << endl;
    }

    // Calculate total value of that item
    float getTotalValue() const {
        return price * quantity;
    }

    // Destructor
    ~Item() {
        cout << "[Item destroyed] ID=" << id << endl;
    }
};

// ================ DERIVED CLASS 1 ================
class Electronics : public Item {
private:
    int warrantyYears;
    float powerUsage;

public:
    // Constructor (calls base constructor + sets derived members)
    Electronics(string name_, int id_, float price_, int quantity_,
                int warranty_, float power_)
        : Item(name_, id_, price_, quantity_) {
        warrantyYears = warranty_;
        powerUsage = power_;
    }

    // Display electronics details
    void displayDetails() const {
        cout << "Category: Electronics" << endl;
        Item::display();
        cout << "  WarrantyYears: " << warrantyYears << endl;
        cout << "  PowerUsage(W): " << powerUsage << endl;
    }

    // Destructor
    ~Electronics() {
        cout << "[Electronics destroyed]" << endl;
```

```
    }
};

// ================ DERIVED CLASS 2 ================
class Grocery : public Item {
private:
    string expiryDate;
    float weight;

public:
    // Constructor
    Grocery(string name_, int id_, float price_, int quantity_,
            string expiry_, float weight_)
        : Item(name_, id_, price_, quantity_) {
        expiryDate = expiry_;
        weight = weight_;
    }

    // Display grocery details
    void displayDetails() const {
        cout << "Category: Grocery" << endl;
        Item::display();
        cout << "  ExpiryDate: " << expiryDate << endl;
        cout << "  Weight(kg): " << weight << endl;
    }

    // Destructor
    ~Grocery() {
        cout << "[Grocery destroyed]" << endl;
    }
};

// ================ MAIN FUNCTION ================
int main() {
    cout << "Enter total number of items: ";
    int n;
    cin >> n;

    if (n <= 0) {
        cout << "Invalid number. Exiting." << endl;
        return 0;
    }
```

```
    Item** items = new Item*[n]; // Array of pointers to Item
    int* types = new int[n];     // 1 = Electronics, 2 = Grocery

    for (int i = 0; i < n; ++i) {
        cout << "\nItem " << (i + 1)
             << " - category (1=Electronics, 2=Grocery): ";
        int cat;
        cin >> cat;

        string name;
        int id;
        float price;
        int qty;

        cout << "  Name (one word): ";
        cin >> name;
        cout << "  ID: ";
        cin >> id;
        cout << "  Price: ";
        cin >> price;
        cout << "  Quantity: ";
        cin >> qty;

        if (cat == 1) {
            int warranty;
            float power;
            cout << "  WarrantyYears: ";
            cin >> warranty;
            cout << "  PowerUsage(W): ";
            cin >> power;

            items[i] = new Electronics(name, id, price, qty, warranty,
power);
            types[i] = 1;
        } else {
            string expiry;
            float weight;
            cout << "  ExpiryDate (YYYY-MM-DD): ";
            cin >> expiry;
            cout << "  Weight(kg): ";
            cin >> weight;
```

```
            items[i] = new Grocery(name, id, price, qty, expiry,
weight);
            types[i] = 2;
        }
    }

    cout << "\n--- Inventory Details ---" << endl;
    float total = 0.0f;

    for (int i = 0; i < n; ++i) {
        cout << "\nItem " << (i + 1) << ":" << endl;

        if (types[i] == 1) {
            static_cast<Electronics*>(items[i])->displayDetails();
        } else {
            static_cast<Grocery*>(items[i])->displayDetails();
        }

        cout << "  Total value: " << items[i]->getTotalValue() << endl;
        total += items[i]->getTotalValue();
    }

    cout << "\nTotal inventory value: " << total << endl;

    // Clean up memory
    for (int i = 0; i < n; ++i) {
        if (types[i] == 1)
            delete static_cast<Electronics*>(items[i]);
        else
            delete static_cast<Grocery*>(items[i]);
    }

    delete[] items;
    delete[] types;

    cout << "Memory cleaned. Exiting." << endl;
    return 0;
}
```

Output:


PS C:\Users\mahir> cd "c:\Users\mahir\OneDrive\Desktop\Su
Enter total number of items: 1
Item 1 - category (1=Electronics, 2=Grocery): 1
Name (one word): laptop
ID: 102
Price: 345
Quantity: 3
WarrantyYears: 1
PowerUsage(W): 3
--- Inventory Details ---
Item 1:
Category: Electronics
Name: laptop
ID: 102
Price: 345
Quantity: 3
WarrantyYears: 1
PowerUsage(W): 3
Total value: 1035
Total inventory value: 1035
[Electronics destroyed]
[Item destroyed] ID=102
Memory cleaned. Exiting.

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.

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

// ================= BASE CLASS =================
class Employee {
private:
    string name;
    int id;
    float baseSalary;

protected:
    float bonus; // accessible by derived classes

public:
    // Simple parameterized constructor
    Employee(string name_, int id_, float baseSalary_) {
        name = name_;
        id = id_;
        baseSalary = baseSalary_;
        bonus = 0;
    }

    // Getter for baseSalary (for derived classes)
    float getBaseSalary() const {
        return baseSalary;
    }

    // Virtual function to calculate bonus
    virtual void calculateBonus() {
        bonus = 0; // base class default
    }

    // Virtual function to display employee info
    virtual void display() {
        cout << "Name: " << name << endl;
        cout << "ID: " << id << endl;
```

```
        cout << "Base Salary: " << fixed << setprecision(2) <<
baseSalary << endl;
        cout << "Bonus: " << fixed << setprecision(2) << bonus << endl;
        cout << "Total Salary: " << fixed << setprecision(2) <<
(baseSalary + bonus) << endl;
    }

    // Virtual destructor for safe cleanup
    virtual ~Employee() {
        cout << "[Employee destroyed]" << endl;
    }
};

// ================= DERIVED CLASS 1 =================
class Manager : public Employee {
public:
    // Constructor
    Manager(string name_, int id_, float baseSalary_) : Employee(name_,
id_, baseSalary_) {}

    // Override bonus calculation
    void calculateBonus() override {
        bonus = 0.4 * getBaseSalary(); // 40% bonus
    }

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

// ================= DERIVED CLASS 2 =================
class Developer : public Employee {
public:
    // Constructor
    Developer(string name_, int id_, float baseSalary_) :
Employee(name_, id_, baseSalary_) {}

    // Override bonus calculation
    void calculateBonus() override {
        bonus = 0.25 * getBaseSalary(); // 25% bonus
    }
```

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

// ================ MAIN FUNCTION ================
int main() {
    int n;
    cout << "Enter number of employees: ";
    cin >> n;

    if (n <= 0) {
        cout << "Invalid number. Exiting." << endl;
        return 0;
    }

    Employee** employees = new Employee*[n]; // array of Employee
pointers

    for (int i = 0; i < n; ++i) {
        cout << "\nEmployee " << (i + 1) << " type (1=Manager,
2=Developer): ";
        int type;
        cin >> type;

        string name;
        int id;
        float salary;

        cout << "Enter name: ";
        cin >> name;
        cout << "Enter ID: ";
        cin >> id;
        cout << "Enter base salary: ";
        cin >> salary;

        if (type == 1) {
            employees[i] = new Manager(name, id, salary);
        } else {
            employees[i] = new Developer(name, id, salary);
```

```
        }

        employees[i]->calculateBonus();
    }

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

    // Delete dynamically allocated objects
    for (int i = 0; i < n; ++i) {
        delete employees[i];
    }

    delete[] employees;

    cout << "\nMemory cleaned. Exiting." << endl;
    return 0;
}
```

Output:

```
PS C:\Users\mahir> cd "c:\Users\mahir\OneDrive\Deskto
Enter number of employees: 1

Employee 1 type (1=Manager, 2=Developer): 1
Enter name: mahi
Enter ID: 233
Enter base salary: 13467

=== Employee Payroll ===

--- Manager Details ---
Name: mahi
ID: 233
Base Salary: 13467.00
Bonus: 5386.80
Total Salary: 18853.80
[Employee destroyed]

Memory cleaned. Exiting.
```

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

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

// ======================== INVENTORY SYSTEM ========================
class Item {
private:
    string name;
    int id;
    float price;

protected:
    int quantity;

public:
    Item(string name_, int id_, float price_, int quantity_) {
        name = name_;
        id = id_;
        price = price_;
        quantity = quantity_;
    }

    virtual void display() const {
        cout << "Name: " << name
             << ", ID: " << id
             << ", Price: $" << fixed << setprecision(2) << price
             << ", Quantity: " << quantity;
    }

    float getTotalValue() const { return price * quantity; }

    virtual ~Item() {
        cout << "[Item destroyed] ID=" << id << endl;
    }
};

class Electronics : public Item {
private:
    int warrantyYears;
    float powerUsage;
```

```
public:
    Electronics(string name_, int id_, float price_, int quantity_, int
warranty_, float power_)
        : Item(name_, id_, price_, quantity_) {
        warrantyYears = warranty_;
        powerUsage = power_;
    }

    void display() const override {
        cout << "\n--- Electronics ---\n";
        Item::display();
        cout << ", Warranty: " << warrantyYears << " years"
             << ", Power: " << powerUsage << "W\n";
    }

    ~Electronics() { cout << "[Electronics destroyed]\n"; }
};

class Grocery : public Item {
private:
    string expiryDate;
    float weight;

public:
    Grocery(string name_, int id_, float price_, int quantity_, string
expiry_, float weight_)
        : Item(name_, id_, price_, quantity_) {
        expiryDate = expiry_;
        this->weight = weight_;
    }

    void display() const override {
        cout << "\n--- Grocery ---\n";
        Item::display();
        cout << ", Expiry: " << expiryDate
             << ", Weight: " << weight << "kg\n";
    }

    ~Grocery() { cout << "[Grocery destroyed]\n"; }
};

// ======================== PAYROLL SYSTEM ========================
class Employee {
```

```
private:
    string name;
    int id;
    float baseSalary;

protected:
    float bonus;

public:
    Employee(string name_, int id_, float baseSalary_) {
        name = name_;
        id = id_;
        baseSalary = baseSalary_;
        bonus = 0;
    }

    float getBaseSalary() const { return baseSalary; }
    virtual float getTotalSalary() const { return baseSalary + bonus; }

    virtual void calculateBonus() { bonus = 0; }

    virtual void display() const {
        cout << "Name: " << name
             << ", ID: " << id
             << ", Base Salary: $" << fixed << setprecision(2) <<
baseSalary
             << ", Bonus: $" << bonus
             << ", Total: $" << (baseSalary + bonus) << endl;
    }

    virtual ~Employee() { cout << "[Employee destroyed]\n"; }
};

class Manager : public Employee {
public:
    Manager(string name_, int id_, float baseSalary_)
        : Employee(name_, id_, baseSalary_) {}

    void calculateBonus() override { bonus = 0.4 * getBaseSalary(); }

    void display() const override {
        cout << "\n--- Manager ---\n";
        Employee::display();
```

```
    }
};

class Developer : public Employee {
public:
    Developer(string name_, int id_, float baseSalary_)
        : Employee(name_, id_, baseSalary_) {}

    void calculateBonus() override { bonus = 0.25 * getBaseSalary(); }

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

// ======================== MENU-DRIVEN EMPLOYEES
========================
class FullTimeEmployee : public Employee {
public:
    FullTimeEmployee(string name_, int id_, float salary_)
        : Employee(name_, id_, salary_) {}

    void display() const override {
        cout << "\n--- Full-Time Employee ---\n";
        Employee::display();
    }

    ~FullTimeEmployee() { cout << "[FullTimeEmployee destroyed]\n"; }
};

class PartTimeEmployee : public Employee {
private:
    float hoursWorked;
    float hourlyRate;

public:
    PartTimeEmployee(string name_, int id_, float rate, float hours)
        : Employee(name_, id_, rate * hours) {
        hoursWorked = hours;
        hourlyRate = rate;
    }
```

```
    void display() const override {
        cout << "\n--- Part-Time Employee ---\n";
        Employee::display();
        cout << "Hours Worked: " << hoursWorked
             << ", Hourly Rate: $" << fixed << setprecision(2) <<
hourlyRate << endl;
    }

    ~PartTimeEmployee() { cout << "[PartTimeEmployee destroyed]\n"; }
};

// ======================== HELPER FUNCTIONS ========================
void clearInput() {
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}

int getIntInput(const string& prompt) {
    int val;
    while (true) {
        cout << prompt;
        if (cin >> val && val >= 0) return val;
        cout << "Invalid input, try again.\n";
        clearInput();
    }
}

float getFloatInput(const string& prompt) {
    float val;
    while (true) {
        cout << prompt;
        if (cin >> val && val >= 0) return val;
        cout << "Invalid input, try again.\n";
        clearInput();
    }
}

// ======================== MAIN MENU ========================
int main() {
    int choice;
    do {
        cout << "\n===== Unified Management System =====\n";
        cout << "1. Inventory System\n";
```

```
        cout << "2. Payroll System\n";
        cout << "3. Menu-Driven Employee System\n";
        cout << "4. Exit\n";
        choice = getIntInput("Enter choice: ");

        if (choice == 1) {
            int n = getIntInput("Number of items: ");
            Item** items = new Item*[n];
            float totalInventoryValue = 0;

            for (int i = 0; i < n; ++i) {
                int t;
                while (true) {
                    t = getIntInput("Item type (1=Electronics,
2=Grocery): ");
                    if (t == 1 || t == 2) break;
                    cout << "Please enter 1 or 2.\n";
                }

                string name; int id; float price; int qty;
                cout << "Name: "; cin >> name;
                id = getIntInput("ID: ");
                price = getFloatInput("Price: ");
                qty = getIntInput("Quantity: ");

                if (t == 1) {
                    int w = getIntInput("Warranty years: ");
                    float p = getFloatInput("Power usage (W): ");
                    items[i] = new Electronics(name,id,price,qty,w,p);
                } else {
                    string exp; cout << "Expiry Date (YYYY-MM-DD): ";
cin >> exp;
                    float w = getFloatInput("Weight (kg): ");
                    items[i] = new Grocery(name,id,price,qty,exp,w);
                }
            }

            cout << "\n--- Inventory Details ---\n";
            for (int i = 0; i < n; ++i) {
                items[i]->display();
                totalInventoryValue += items[i]->getTotalValue();
            }
```

```
            cout << "\nTotal Inventory Value: $" << fixed <<
setprecision(2) << totalInventoryValue << endl;

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

        else if (choice == 2) {
            int n = getIntInput("Number of employees: ");
            Employee** emp = new Employee*[n];
            float totalPayroll = 0;

            for (int i = 0; i < n; ++i) {
                int t;
                while (true) {
                    t = getIntInput("Employee type (1=Manager,
2=Developer): ");
                    if (t == 1 || t == 2) break;
                    cout << "Please enter 1 or 2.\n";
                }

                string name; int id; float salary;
                cout << "Name: "; cin >> name;
                id = getIntInput("ID: ");
                salary = getFloatInput("Base Salary: ");
                if (t == 1) emp[i] = new Manager(name,id,salary);
                else emp[i] = new Developer(name,id,salary);
                emp[i]->calculateBonus();
                totalPayroll += emp[i]->getTotalSalary();
            }

            cout << "\n--- Payroll Details ---\n";
            for(int i=0;i<n;++i) emp[i]->display();
            cout << "\nTotal Payroll: $" << fixed << setprecision(2) <<
totalPayroll << endl;

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

        else if (choice == 3) {
            const int MAX = 100;
            Employee* emps[MAX];
```

```
            int count = 0;
            int subchoice;
            do {
                cout << "\n1. Add Full-Time\n2. Add Part-Time\n3.
Display Employees\n4. Back\n";
                subchoice = getIntInput("Choice: ");

                if(subchoice==1) {
                    string name; int id; float salary;
                    cout << "Name: "; cin >> name;
                    id = getIntInput("ID: ");
                    salary = getFloatInput("Salary: ");
                    emps[count++] = new
FullTimeEmployee(name,id,salary);
                }
                else if(subchoice==2) {
                    string name; int id; float rate,hours;
                    cout << "Name: "; cin >> name;
                    id = getIntInput("ID: ");
                    rate = getFloatInput("Hourly rate: ");
                    hours = getFloatInput("Hours worked: ");
                    emps[count++] = new
PartTimeEmployee(name,id,rate,hours);
                }
                else if(subchoice==3) {
                    cout << "\n--- Employee List ---\n";
                    float totalSalary = 0;
                    for(int i=0;i<count;++i) {
                        emps[i]->display();
                        totalSalary += emps[i]->getTotalSalary();
                    }
                    cout << "\nTotal Salary for all employees: $" <<
fixed << setprecision(2) << totalSalary << endl;
                }
            } while(subchoice!=4);

            for(int i=0;i<count;++i) delete emps[i];
        }

        else if(choice != 4) {
            cout << "Invalid choice. Try again.\n";
        }
```

```
    } while(choice != 4);

    cout << "\nExiting Unified Management System... Goodbye!\n";
    return 0;
}
```

Output:

```
===== Unified Management System =====
1. Inventory System
2. Payroll System
3. Menu-Driven Employee System
4. Exit
Enter choice: 1
Number of items: 1
Enter choice: 1
Enter choice: 1
Number of items: 1
Item type (1=Electronics, 2=Grocery): 1
Name: lp
ID: 101
Price: 98000
Quantity: 3
Warranty years: 34
Power usage (W): 255

--- Inventory Details ---

--- Electronics ---
Name: lp, ID: 101, Price: $98000.00, Quantity: 3, Warranty: 34 years, Power: 255.00W

Total Inventory Value: $294000.00
[Electronics destroyed]
[Item destroyed] ID=101

===== Unified Management System =====
1. Inventory System
2. Payroll System
3. Menu-Driven Employee System
4. Exit
Enter choice: 4

Exiting Unified Management System... Goodbye!
```