

## 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.

Microsoft Visual Studio IDE showing the code for `Item.h` and the Solution Explorer.

**Code Editor:**

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

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

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

    float getTotalValue() {
        return price * quantity;
    }

    ~Item() {
        cout << "Destroying Item: " << name << endl;
    }
};

class Electronics : public Item {
private:
    int warrantyYears;
    float powerUsage;
public:
    Electronics(string n, int i, float p, int q, int w, float pu)
        : Item(n, i, p, q), warrantyYears(w), powerUsage(pu) {}

    void displayDetails() {
        cout << "This is an Electronics Item ---" << endl;
    }
};

68 %
```

**Solution Explorer:**

- Solution 'Basic' (1 of 1 project)
  - Basic
    - References
    - External Dependencies
    - Header Files
    - Resource Files
    - Source Files
      - Bank.cpp
      - Basic.cpp
      - ComplexNumber.cpp
      - Employee.cpp
      - InventoryManagement.cpp
      - Library.cpp
      - Rectangle.cpp
      - Student.cpp

Microsoft Visual Studio IDE showing the code for `Item.h` and the Solution Explorer.

**Code Editor:**

```
int warrantyYears;
float powerUsage;
public:
    Electronics(string n, int i, float p, int q, int w, float pu)
        : Item(n, i, p, q), warrantyYears(w), powerUsage(pu) {}

    void displayDetails() {
        cout << "This is an Electronics Item ---" << endl;
        display();
        cout << "Warranty (years): " << warrantyYears << endl;
        cout << "Power Usage (Kwh/c): " << powerUsage << endl;
        cout << "Total Value: " << getTotalValue() << endl;
    }

    ~Electronics() {
        cout << "Cleaning up Electronics item...\n";
    }
};

class Grocery : public Item {
private:
    string expiryDate;
    float weight;
public:
    Grocery(string n, int i, float p, int q, string ed, float w)
        : Item(n, i, p, q), expiryDate(ed), weight(w) {}

    void displayDetails() {
        cout << "This is a Grocery Item ---" << endl;
        display();
        cout << "Expiry Date: " << expiryDate << endl;
        cout << "Weight (kg): " << weight << endl;
        cout << "Total Value: " << getTotalValue() << endl;
    }

    ~Grocery() {
        cout << "Cleaning up Grocery item...\n";
    }
};

int main() {
    int totalItems;
    cout << "Enter total number of items: ";
    cin >> totalItems;
    Item items[100];
}
```

**Solution Explorer:**

- Solution 'Basic' (1 of 1 project)
  - Basic
    - References
    - External Dependencies
    - Header Files
    - Resource Files
    - Source Files
      - Bank.cpp
      - Basic.cpp
      - ComplexNumber.cpp
      - Employee.cpp
      - InventoryManagement.cpp
      - Library.cpp
      - Rectangle.cpp
      - Student.cpp

```

1 // Main function
2 int main() {
3     int totalItems;
4     cout << "Enter total number of items: ";
5     cin >> totalItems;
6     cout << endl;
7     int itemCount = 0;
8     float totalInventoryValue = 0.0;
9
10    for (int i = 0; i < totalItems; i++) {
11        cout << "Enter category of item " << (i + 1) << endl;
12        cout << "1. Electronics\n2. Grocery" << endl;
13        cin >> choice;
14
15        if (choice == 1) {
16            string name;
17            int id, quantity, warranty;
18            float price, power;
19            cout << "Enter Name, ID, Price, Quantity, WarrantyYears, PowerUsage: \n";
20            cin >> name >> id >> price >> quantity >> warranty >> power;
21            item(itemCount) = new Electronics(name, id, price, quantity, warranty, power);
22        } else if (choice == 2) {
23            string name, expiry;
24            int id, qty;
25            float weight;
26            cout << "Enter Name, ID, Price, Quantity, ExpiryDate, Weight: \n";
27            cin >> name >> id >> price >> qty >> expiry >> weight;
28            item(itemCount) = new Grocery(name, id, price, qty, expiry, weight);
29        } else {
30            cout << "Invalid choice. Skipping item." << endl;
31        }
32    }
33
34    cout << "\n----- DESTROY DETAILS -----";
35    for (int i = 0; i < itemCount; i++) {
36        cout << endl;
37        cout << "Electronics " << static_cast<Electronics*>(item[i]);
38        cout << "Grocery " << static_cast<Grocery*>(item[i]);
39
40        if (*e) {
41            e->displayDetails();
42        } else if (*g) {
43            g->displayDetails();
44        }
45        totalInventoryValue += item[i]->getTotalValue();
46
47        cout << "\nTotal Inventory Value: " << totalInventoryValue << endl;
48    }
49
50    for (int i = 0; i < itemCount; i--) {
51        delete item[i];
52    }
53
54    cout << "All items deleted successfully.\n";
55    return 0;
56}

```

```

Microsoft Visual Studio Debug X + v
Enter total number of items: 2
Enter category of item 1:
1. Electronics
2. Grocery
Choice: 1
Enter Name, ID, Price, Quantity, WarrantyYears, PowerUsage:
AC
101
47500
2
2
15
Enter category of item 2:
1. Electronics
2. Grocery
Choice: 2
Enter Name, ID, Price, Quantity, ExpiryDate, Weight:
Rice
102
105
10
12-2025
10
=====
INVENTORY DETAILS =====
--- Electronics Item ---
Item Name: AC
Item ID: 101
Price: 47500
Quantity: 2
Warranty (years): 2
Power Usage (Watts): 15
Total Value: 95000

--- Electronics Item ---
Item Name: Rice
Item ID: 102
Price: 105
Quantity: 10
Warranty (years): 350183168
Power Usage (Watts): 6.02558e-43
Total Value: 1050
Total Inventory Value: 96050
Destroying Item: AC
Destroying Item: Rice
All items deleted successfully.

C:\Users\vipul\source\repos\Basic\x64\Debug\Basic.exe (process 18584) exited with code 0 (0x0).

```

## 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).

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.

The screenshot shows the Microsoft Visual Studio IDE interface with the following details:

- Title Bar:** File, Edt, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search.
- Toolbar:** Standard icons for file operations like Open, Save, Print, etc.
- Status Bar:** Shows the current process (16920) and the date (08-10-2023).
- Code Editor:** The main window displays the `EmployeeSalary.h` file. The code defines a base class `Employee` with private members `name`, `id`, and `baseSalary`, and protected members `bonus` and `getBaseSalary()`. It also includes a constructor, a virtual `calculateBonus()` method, a `display()` method, and a `~Employee()` destructor. A derived class `Manager` inherits from `Employee` and overrides the `calculateBonus()` and `display()` methods. The code uses C++11 features like `#include <iostream>` and `using namespace std;`.
- Solution Explorer:** Shows the project structure with files like `EmployeeSalary.h`, `InventoryManagement.cpp`, `Library.cpp`, `ComplexNumber.cpp`, `Employee.cpp`, `Bank.cpp`, `Student.cpp`, `Rectangle.cpp`, and `Basic.cpp`.
- Toolbars:** Standard, Debug, and GitHub Copilot.
- Status Bar:** Shows the current line (Ln: 118), character (Ch: 1), and other status information.

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search Basic
Process: [16920] Basic.exe Lifecycle Events Thread Stack Frame Application Insights GitHub Copilot
EmployeeSalaryCalculation.cpp InventoryManagement.cpp Library.cpp ComplexNumber.cpp Employee.cpp Bank.cpp Student.cpp Rectangle.cpp Basic.cpp
Basic
43
44
45 void display() override {
46     cout << "\n--- Manager Details ---" << endl;
47     Employee::display();
48 }
49
50 ~Manager() {
51     cout << "Cleaning up Manager...\n";
52 }
53
54 class Developer : public Employee {
55 public:
56     Developer(string n, int i, float s)
57         : Employee(n, i, s) {}
58
59     void calculateBonus() override {
60         bonus = 0.25f * getBaseSalary();
61     }
62
63     void display() override {
64         cout << "\n--- Developer Details ---" << endl;
65         Employee::display();
66     }
67
68 ~Developer() {
69     cout << "Cleaning up Developer...\n";
70 }
71
72
73 int main() {
74     int totalEmployees;
75     cout << "Enter total number of employees: ";
76     cin >> totalEmployees;
77
78     Employee* empList[100];
79     int empCount = 0;
80
81     for (int i = 0; i < totalEmployees; i++) {
82         int choice;
83         cout << "Enter type of employee " << (i + 1) << ":" << endl;
84         cout << "1. Manager\n2. Developer\nChoice: ";
85         cin >> choice;
86
87         string name;
88         int id;
89         float salary;
90
91         cout << "Enter Name, ID, Base Salary: ";
92         cin >> name >> id >> salary;
93
94         if (choice == 1)
95             empList[empCount++] = new Manager(name, id, salary);
96         else if (choice == 2)
97             empList[empCount++] = new Developer(name, id, salary);
98         else
99             cout << "Invalid choice! Skipping.\n";
100
101     cout << "\n===== EMPLOYEE DETAILS =====\n";
102
103     for (int i = 0; i < empCount; i++) {
104         empList[i]->calculateBonus();
105         empList[i]->display();
106     }
107
108     for (int i = 0; i < empCount; i++) {
109         delete empList[i];
110     }
111
112     cout << "All employees deleted successfully.\n";
113     return 0;
114 }
```

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search Basic
Process: [16920] Basic.exe Lifecycle Events Thread Stack Frame Application Insights GitHub Copilot
EmployeeSalaryCalculation.cpp InventoryManagement.cpp Library.cpp ComplexNumber.cpp Employee.cpp Bank.cpp Student.cpp Rectangle.cpp Basic.cpp
Basic
76
77     cout << "Enter total number of employees: ";
78     cin >> totalEmployees;
79
80     Employee* empList[100];
81     int empCount = 0;
82
83     for (int i = 0; i < totalEmployees; i++) {
84         int choice;
85         cout << "Enter type of employee " << (i + 1) << ":" << endl;
86         cout << "1. Manager\n2. Developer\nChoice: ";
87         cin >> choice;
88
89         string name;
90         int id;
91         float salary;
92
93         cout << "Enter Name, ID, Base Salary: ";
94         cin >> name >> id >> salary;
95
96         if (choice == 1)
97             empList[empCount++] = new Manager(name, id, salary);
98         else if (choice == 2)
99             empList[empCount++] = new Developer(name, id, salary);
100         else
101             cout << "Invalid choice! Skipping.\n";
102
103     cout << "\n===== EMPLOYEE DETAILS =====\n";
104
105     for (int i = 0; i < empCount; i++) {
106         empList[i]->calculateBonus();
107         empList[i]->display();
108     }
109
110     for (int i = 0; i < empCount; i++) {
111         delete empList[i];
112     }
113
114     cout << "All employees deleted successfully.\n";
115     return 0;
116 }
```

```

Microsoft Visual Studio Debug

Enter total number of employees: 2
C:\Users\vipul\source\repos\Basic\x64\Debug\Basic.exe (process 16928) exited with code -1 (0xffffffff).
Enter type of employee 1:
1. Manager
2. Developer
Choice: 1
Enter Name, ID, Base Salary: Vipul
111
65000
Enter type of employee 2:
1. Manager
2. Developer
Choice: 2
Enter Name, ID, Base Salary: Sagar
22
75000
===== EMPLOYEE DETAILS =====
--- Manager Details ---
Employee Name: Vipul
Employee ID: 111
Base Salary: 65000
Bonus: 26000
Total Salary: 91000
--- Developer Details ---
Employee Name: Sagar
Employee ID: 22
Base Salary: 75000
Bonus: 18750
Total Salary: 93750
Cleaning up Manager...
Destroying Employee: Vipul
Cleaning up Developer...
Destroying Employee: Sagar
All employees deleted successfully.

C:\Users\vipul\source\repos\Basic\x64\Debug\Basic.exe (process 13900) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . .

```

3. Title: Menu-Driven Employee Management System using Classes, 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

- 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.

```

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.

EmployeeManagement.cpp
Basic
1 //include <iostream>
2 #include <string>
3 using namespace std;
4
5 class Employee {
6 private:
7     string name;
8     int empID;
9 protected:
10    float salary;
11 public:
12    Employee(string n, int id) : name(n), empID(id), salary(0) {}
13    void displayBasic() { cout << "Name: " << name << ", ID: " << empID << endl; }
14    float getSalary() { return salary; }
15    int getId() { return empID; }
16    virtual void displayDetails() = 0;
17    virtual ~Employee() { cout << "Destroying Employee: " << name << endl; }
18};
19
20 class FullTimeEmployee : public Employee {
21 private:
22    float basicPay;
23    float bonus;
24 public:
25    FullTimeEmployee(string n, int id, float bPay, float b) : Employee(n, id), basicPay(bPay), bonus(b) {}
26    void calculateSalary() { salary = basicPay + bonus; }
27    void displayDetails() override {
28        cout << "----- Full-Time Employee -----" << endl;
29        displayBasic();
30        cout << "Basic Pay: " << basicPay << ", Bonus: " << bonus << endl;
31        cout << "Total Salary: " << salary << endl;
32    }
33    ~FullTimeEmployee() { cout << "Cleaning up Full-Time Employee..." << endl; }
34};
35
36 class PartTimeEmployee : public Employee {
37 private:
38    int hoursWorked;
39    float hourlyRate;
40 public:
41    PartTimeEmployee(string n, int id, int worked, float hRate) : Employee(n, id), hoursWorked(hoursWorked), hourlyRate(hRate) {}
42    void calculateSalary() { salary = hoursWorked * hourlyRate; }
43    void displayDetails() override {
44        cout << "----- Part-Time Employee -----" << endl;
45    }
46    ~PartTimeEmployee() { cout << "Cleaning up Part-Time Employee..." << endl; }
47};
48
49
50 void display() override {
51     cout << "----- Part-Time Employee -----" << endl;
52     displayBasic();
53     cout << "Hours Worked: " << hoursWorked << ", Hourly Rate: " << hourlyRate << endl;
54     cout << "Total Salary: " << salary << endl;
55 }
56
57 ~PartTimeEmployee() { cout << "Cleaning up Part-Time Employee..." << endl; }
58
59
60 int main() {
61     Employee* empList[100];
62     int empCount = 0;
63     int choice;
64
65     do {
66         cout << "\nEnter:\n1.Add Full-Time Employee\n2.Add Part-Time Employee\n3.Display All Employees\n4.Search Employee by ID\n5.Delete Employee by ID\n6.Exit\nChoice: ";
67         cin >> choice;
68
69         if (choice == 1) {
70             string name;
71             int id;
72             float basic, bonus;
73             cout << "Enter Name, ID, Basic Pay, Bonus: ";
74             cin >> name >> id >> basic >> bonus;
75             FullTimeEmployee fte = new FullTimeEmployee(name, id, basic, bonus);
76             fte->calculateSalary();
77             empList[empCount] = fte;
78             empCount++;
79         }
80         else if (choice == 2) {
81             string name;
82             int id;
83             float rate;
84             cout << "Enter Name, ID, Hours Worked, Hourly Rate: ";
85             cin >> name >> id >> hours >> rate;
86             PartTimeEmployee pte = new PartTimeEmployee(name, id, hours, rate);
87             pte->calculateSalary();
88             empList[empCount] = pte;
89             empCount++;
90         }
91         else if (choice == 3) {
92             cout << "----- All Employees -----" << endl;
93             for (int i = 0; i < empCount; i++) {
94                 empList[i]->displayDetails();
95             }
96         }
97         else if (choice == 4) {
98             int searchID;
99             cout << "Enter Employee ID to search: ";
100            cin >> searchID;
101            bool found = false;
102            for (int i = 0; i < empCount; i++) {
103                if (empList[i]->getId() == searchID) {
104                    cout << "Employee found!" << endl;
105                    found = true;
106                }
107            }
108            if (!found) {
109                cout << "Employee not found." << endl;
110            }
111        }
112    } while (choice != 6);
113
114    cout << "Program ended." << endl;
115
116    return 0;
117}

```

Screenshot of a Microsoft Visual Studio IDE session showing the development and execution of a C++ application for Employee Management.

**Code Editor:**

```

40 cout << "---- All Employees ----\n";
41 for (int i = 0; i < empCount; i++)
42     empList[i].displayDetails();
43
44 else if (choice == 4) {
45     int searchID;
46     cout << "Enter Employee ID to search: ";
47     cin >> searchID;
48     bool found = false;
49     for (int i = 0; i < empCount; i++) {
50         if (empList[i].getID() == searchID) {
51             empList[i].displayDetails();
52             found = true;
53         }
54     }
55     if (!found) cout << "Employee not found.\n";
56 }
57 else if (choice == 5) {
58     int delID;
59     cout << "Enter Employee ID to delete: ";
60     cin >> delID;
61     bool found = false;
62     for (int i = 0; i < empCount; i++) {
63         if (empList[i].getID() == delID) {
64             delete empList[i];
65             for (int j = i; j < empCount - 1; j++)
66                 empList[j] = empList[j + 1];
67             empCount--;
68             cout << "Employee deleted.\n";
69             found = true;
70             break;
71         }
72     }
73     if (!found) cout << "Employee not found.\n";
74 }
75 else (choice != 6);
76
77 for (int i = 0; i < empCount; i++)
78     delete empList[i];
79
80 cout << "All employees cleaned up. Exiting program.\n";
81 return 0;
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132

```

**Output Window:**

```

C:\Users\vipul\source\repos\U
Menu:
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
Choice: 1
Enter Name, ID, Basic Pay, Bonus:
Vipul
101
65000
5000

Menu:
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
Choice: 2
Enter Name, ID, Hours Worked, Hourly Rate:
Roshan
102
6
1200

Menu:
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
Choice: 3

---- All Employees ----

---- Full-Time Employee ---
Name: Vipul, ID: 101
Basic Pay: 65000, Bonus: 5000
Total Salary: 70000

---- Part-Time Employee ---
Name: Roshan, ID: 102

```