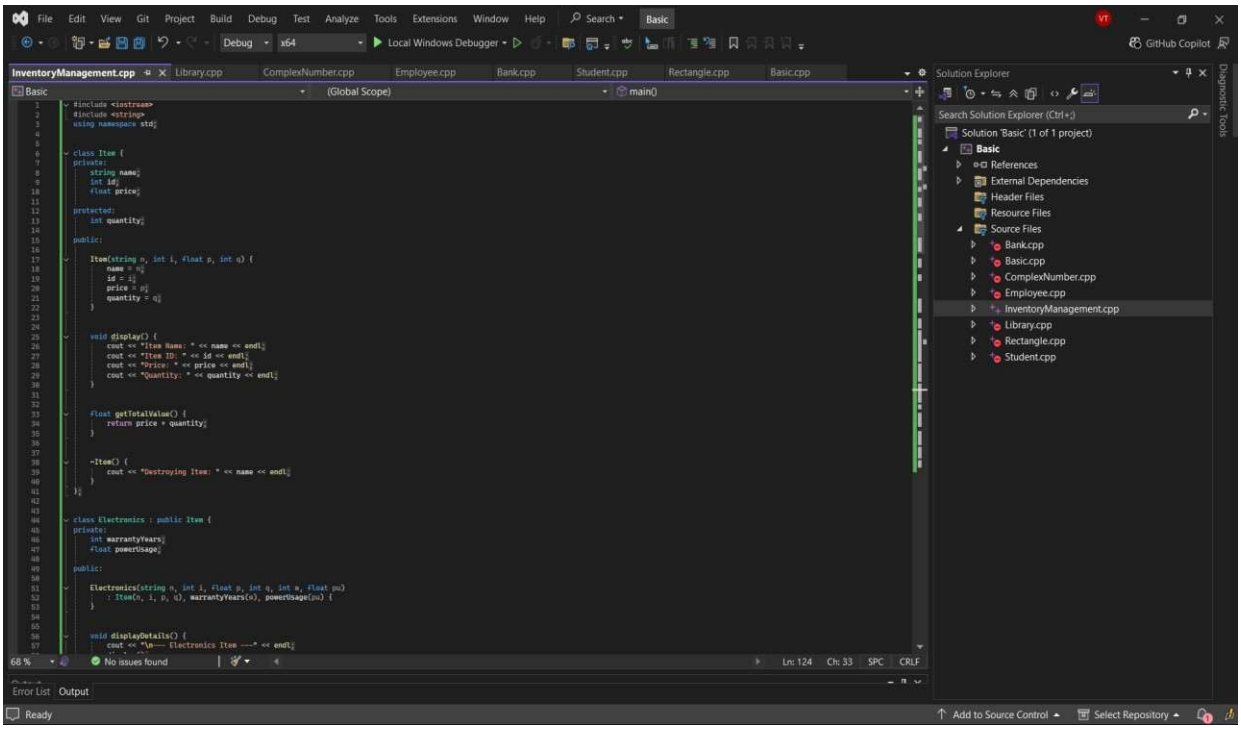
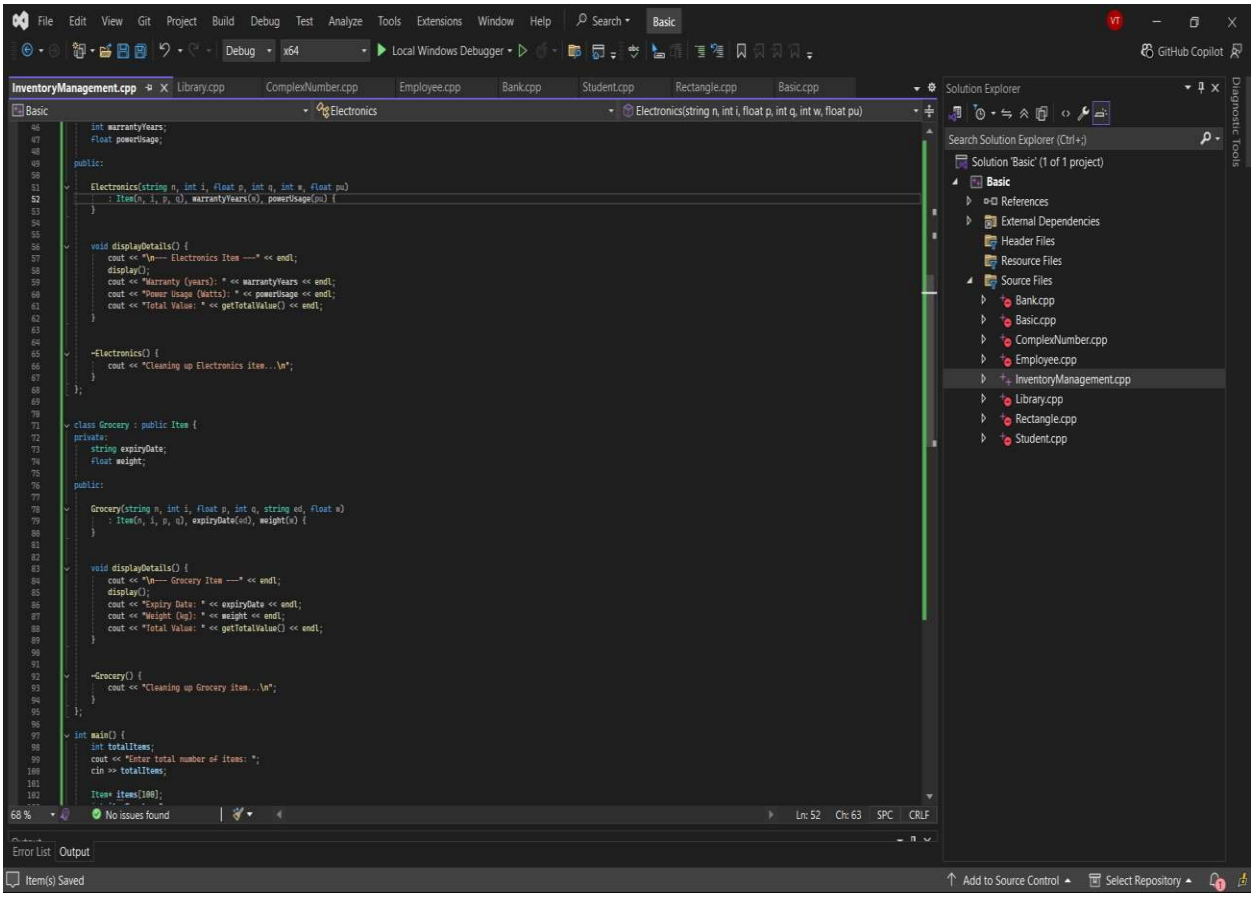
# Assignment 04(Roll no : 107)

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:

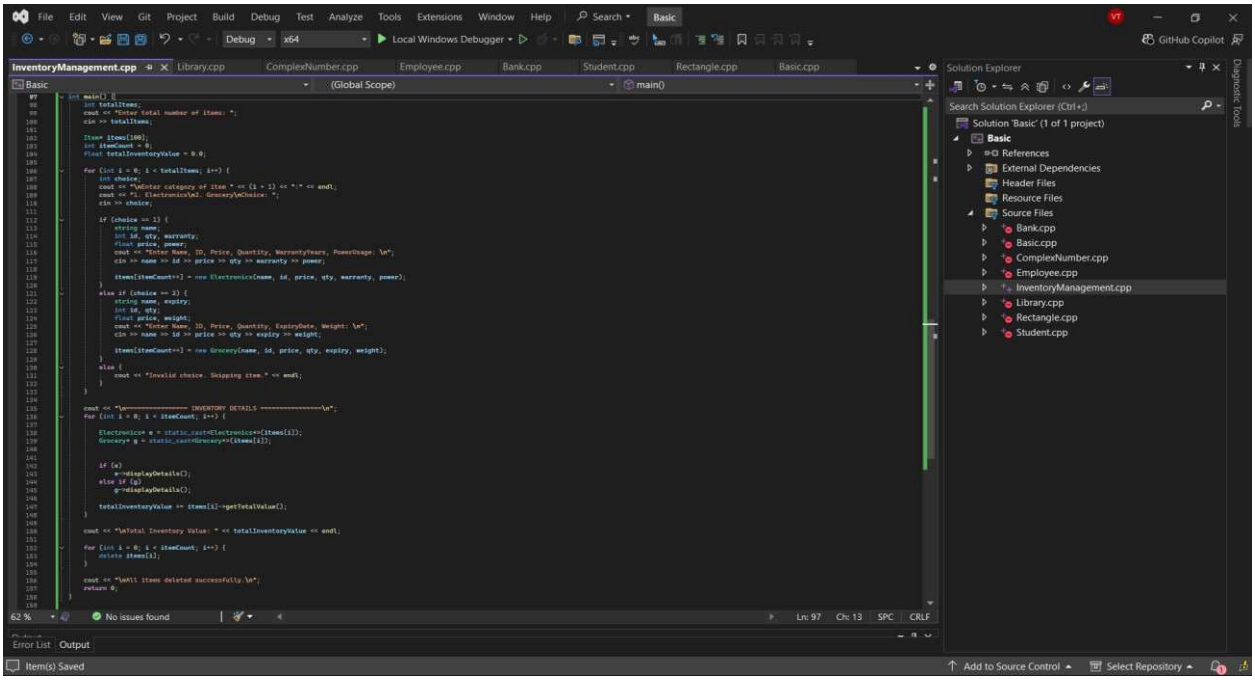
1. Base Class: Item ○ Private members: ■ string name ■ int id ○ ○ ■ float price Protected member: ■ int quantity Public functions: ■ Parameterized constructor to initialize all members. ■ void display() — prints item details. ■ float getTotalValue() — returns price * quantity. ■ Destructor — prints when 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.



```
#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 << "\n--- Electronics Item ---" << endl;
```



```
    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 << "\n--- Electronics Item ---" << endl;
        display();
        cout << "Warranty (years): " << warrantyYears << endl;
        cout << "Power Usage (Watts): " << 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 << "\n--- 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];
```

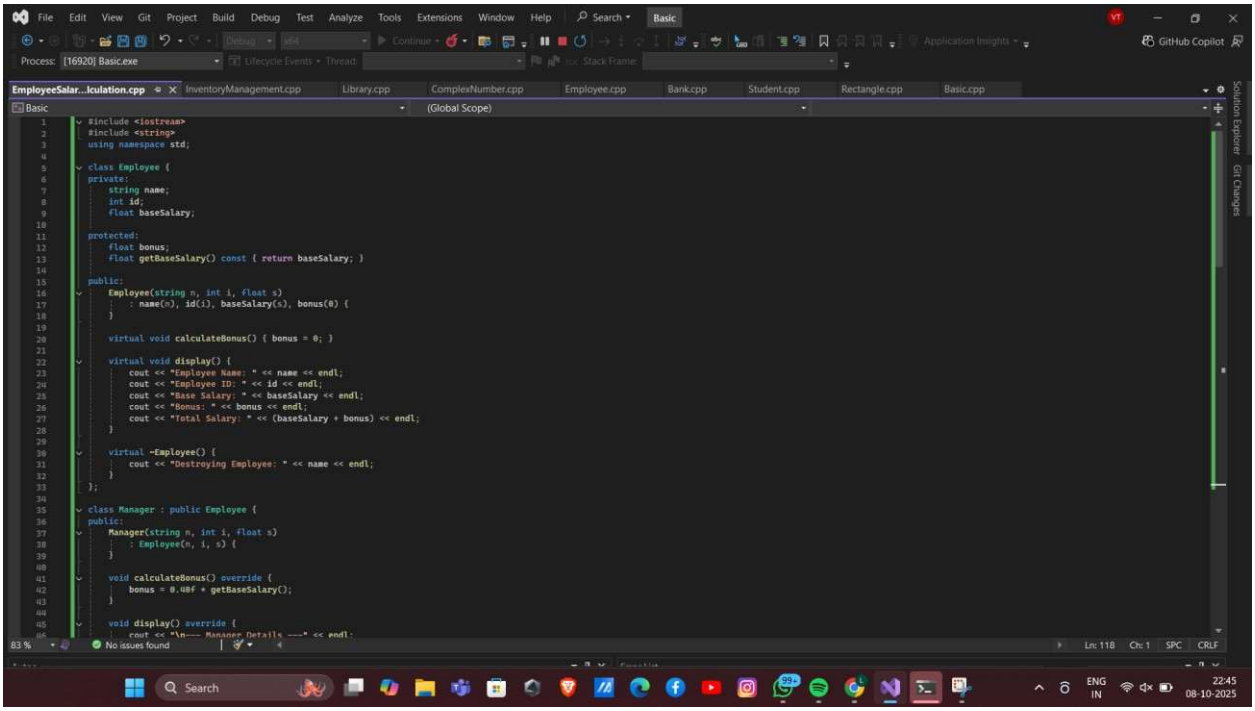


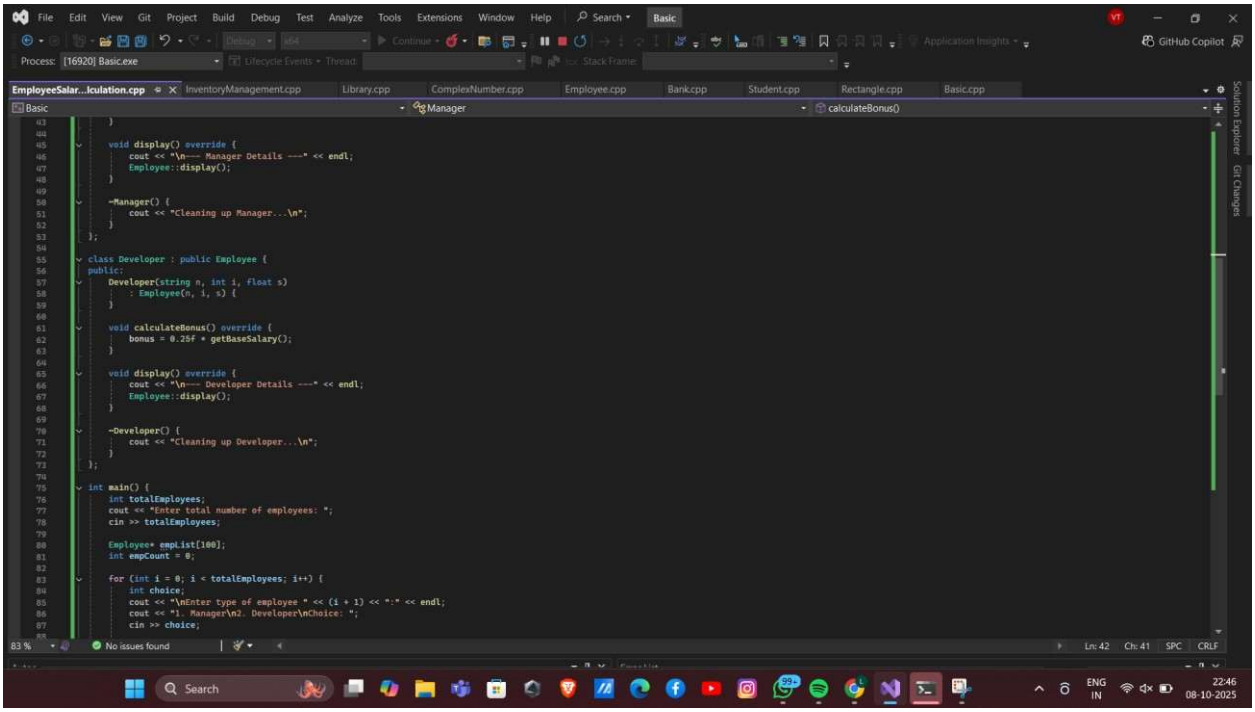
```
Microsoft Visual Studio Debu

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

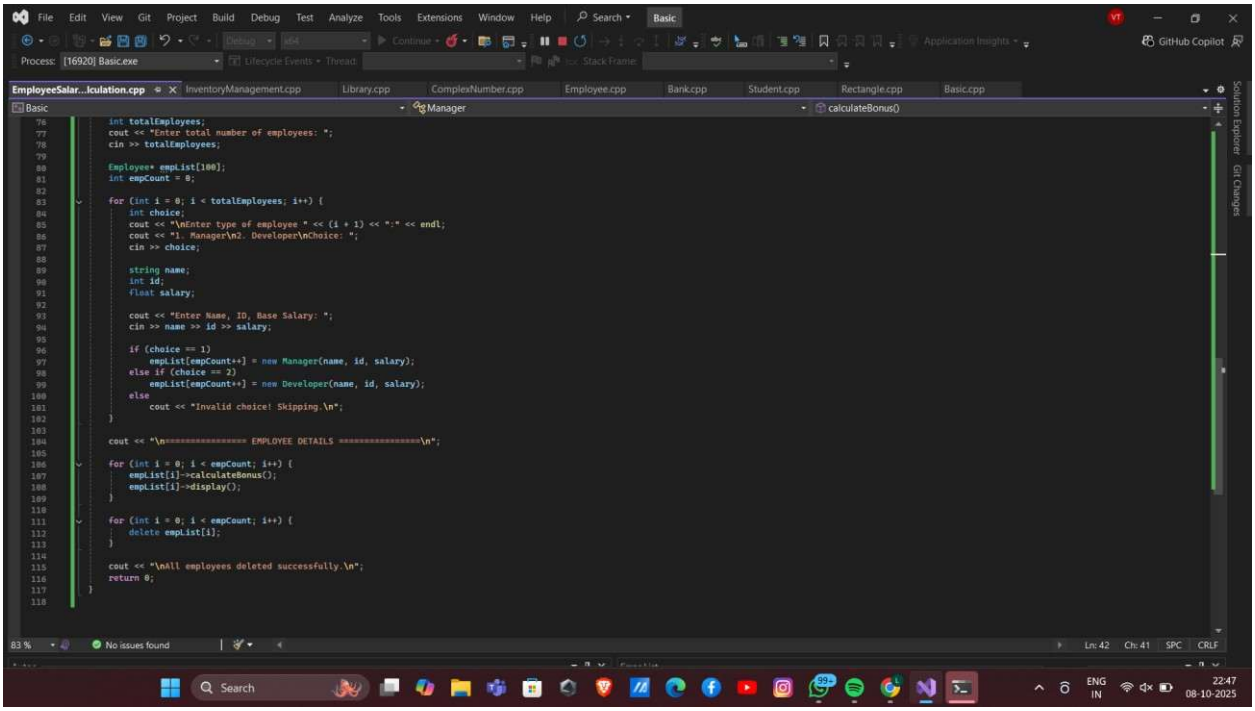
Statement: Design a C++ program to manage employees of a company. Each employee has common details (name, ID, base salary), but different roles (e.g., Manager, Developer) that determine their bonus. You must use classes, inheritance, encapsulation, constructors, destructors, and pointers to: ● Store and display employee information. ● Dynamically allocate memory for employees. ● ● Compute their total salary (base + bonus). 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.

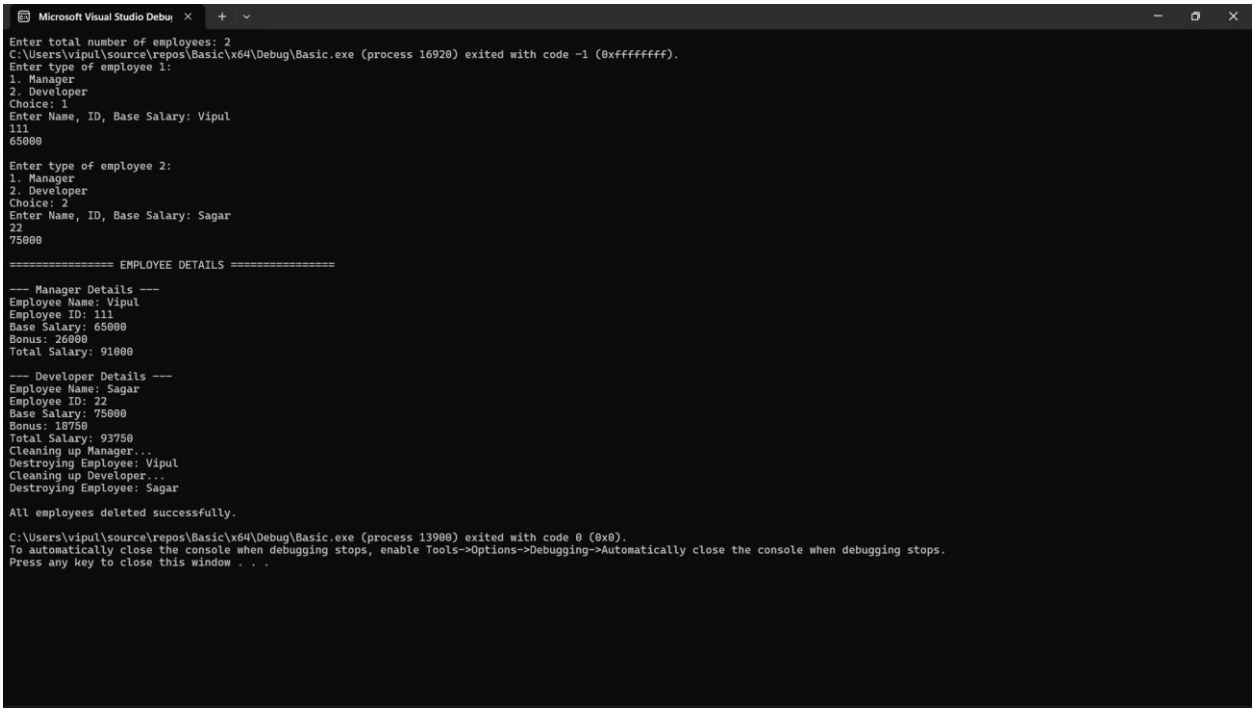




```
    }

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

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

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

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

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

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

int main() {
    int totalEmployees;
    cout << "Enter total number of employees: ";
    cin >> totalEmployees;

    Employee* empList[100];
    int empCount = 0;

    for (int i = 0; i < totalEmployees; i++) {
        int choice;
        cout << "\nEnter type of employee " << (i + 1) << ":" << endl;
        cout << "1. Manager\n2. Developer\nChoice: ";
        cin >> choice;
```



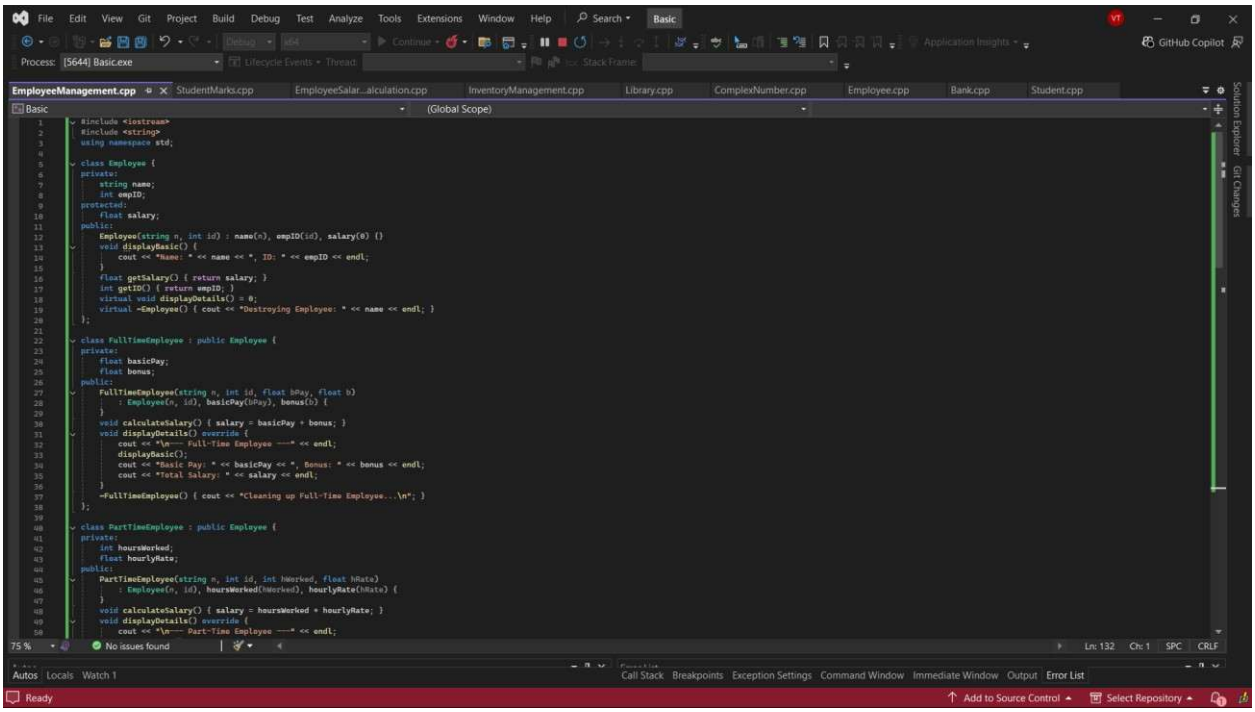
```
    int totalEmployees;
    cout << "Enter total number of employees: ";
    cin >> totalEmployees;

    Employee* empList[100];
    int empCount = 0;

    for (int i = 0; i < totalEmployees; i++) {
        int choice;
        cout << "\nEnter type of employee " << (i + 1) << ":" << endl;
        cout << "1. Manager\n2. Developer\nChoice: ";
        cin >> choice;

        string name;
        int id;
        float salary;

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

        if (choice == 1)
            empList[empCount++] = new Manager(name, id, salary);
        else if (choice == 2)
            empList[empCount++] = new Developer(name, id, salary);
        else
            cout << "Invalid choice! Skipping.\n";
    }

    cout << "\n========== EMPLOYEE DETAILS ==========\n";

    for (int i = 0; i < empCount; i++) {
        empList[i]->calculateBonus();
        empList[i]->display();
    }

    for (int i = 0; i < empCount; i++) {
        delete empList[i];
    }

    cout << "\nAll employees deleted successfully.\n";
    return 0;
}
```

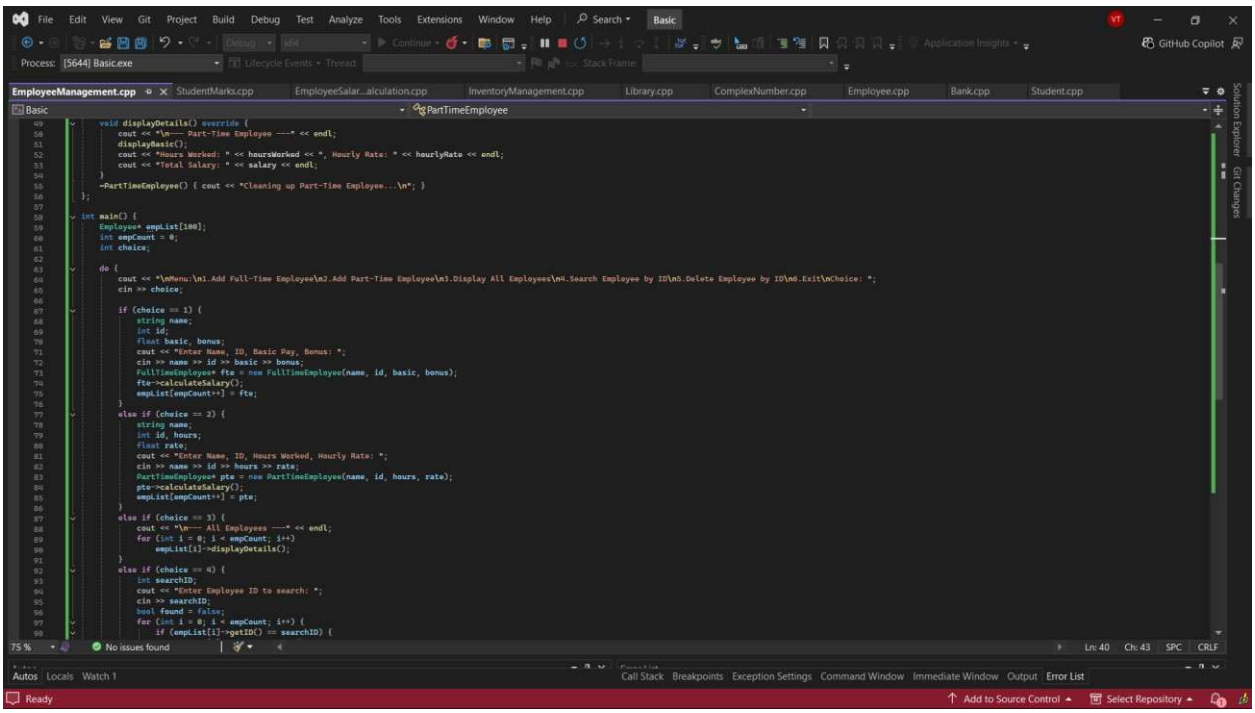


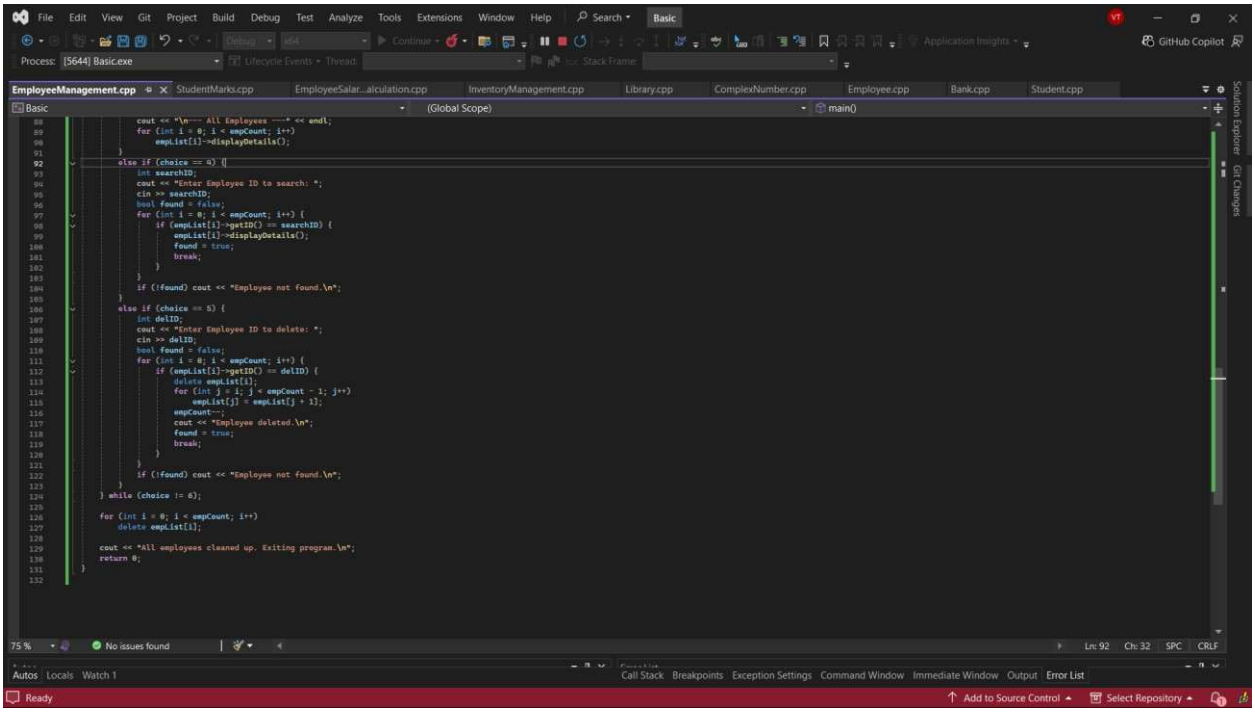

3.Title: Menu-Driven Employee Management System usingClasses, Objects, Inheritance, and Dynamic Memory in C++ Problem Statement Design a Menu-Driven Employee Management System for a company that manages two types of employees:

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.







```
            cout << "\n--- All Employees ---" << endl;
            for (int i = 0; i < empCount; i++)
                empList[i]->displayDetails();
        }
        else if (choice == 4) {
            int searchID;
            cout << "Enter Employee ID to search: ";
            cin >> searchID;
            bool found = false;
            for (int i = 0; i < empCount; i++) {
                if (empList[i]->getID() == searchID) {
                    empList[i]->displayDetails();
                    found = true;
                    break;
                }
            }
            if (!found) cout << "Employee not found.\n";
        }
        else if (choice == 5) {
            int delID;
            cout << "Enter Employee ID to delete: ";
            cin >> delID;
            bool found = false;
            for (int i = 0; i < empCount; i++) {
                if (empList[i]->getID() == delID) {
                    delete empList[i];
                    for (int j = i; j < empCount - 1; j++)
                        empList[j] = empList[j + 1];
                    empCount--;
                    cout << "Employee deleted.\n";
                    found = true;
                    break;
                }
            }
            if (!found) cout << "Employee not found.\n";
        }
    } while (choice != 6);

    for (int i = 0; i < empCount; i++)
        delete empList[i];

    cout << "All employees cleaned up. Exiting program.\n";
    return 0;
}
```

```
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
```