Assignment 2 - ElMenus Food Delivery Management System

Instructions

1. Students will form teams of **2 students** from the same lab group.

2. If more than 2 students submit the assignment, all team members will get **ZERO** for the assignment.

3. Deadline of submission is **Saturday, December 6, 2025, at 11:59 PM**.

4. Submissions will be done through a **Google Form** link that will be posted on Google Classroom.

5. NO late submission is allowed.

6. NO submission through e-mails.

7. In case of Cheating you will get a NEGATIVE GRADE whether you give the code to someone, take the code from someone/internet/AI, or even send it to someone for any reason. Our system uses advanced plagiarism detection algorithms.

8. You must write **clean code** and follow a **good coding style** including choosing **meaningful variable names**.

9. You will develop the needed **.cpp** and **.h** files and should be put in a folder named **Assign2_firstStudentID_SecondStudentID** and compress them to a .zip file with the same folder name. The compressed file would be the file to be delivered. **Failing to abide by this naming convention will result in grades' deduction.**

Task

You have been hired as a **Backend Software Engineer** at **ElMenus**, Egypt's leading food delivery platform. Your task is to develop the **core order management system** using OOP principles.

Using classes, inheritance, polymorphism, aggregation, and dynamic memory management, you will build a system that manages customers, delivery drivers, food items, and orders.

System Components

Enumerations

a. Create `Enum` class `OrderStatus` with

values: `PENDING, PREPARING, OUT_FOR_DELIVERY, DELIVERED, CANCELLED`

b. Create `Enum` class `UserType` with

values: `CUSTOMER, DRIVER`

Class FoodItem

a. Declare **`private`** instance variables: `itemName` (`string`), `price` (`double`), `quantity` (`int`)

b. Create default `constructor`

c. Create parameterized `constructor`

d. Create getters and setters for all attributes

e. Create method **`calculateItemTotal()`**

f. Create method **`displayItem()`** that displays item in format:

Chicken Shawarma x2 @ 45.00 EGP = 90.00 EGP

Class User (Abstract Base Class)

a. Declare **`protected`** instance

variables: **`userId`** (string), **`name`** (string), **`phoneNumber`** (string), **`totalUsers`** (static int)

b. Create default `constructor`

c. Create parameterized `constructor`

d. Declare **pure virtual function `displayInfo()`**

e. Declare **pure virtual function `calculateEarnings()`** that returns double

f. Create static function **`getTotalUsers()`**

g. Create getters for `userId`, name, `phoneNumber`

h. Create virtual `destructor`

Class Customer (Inheritance)

a. Create class **`Customer`** as subclass of **`User`**

b. Declare **`private`** instance variables: **`deliveryAddress`** (string), **`loyaltyPoints`** (int)

c. Create default & parameterized **`constructor`**

d. Override **`displayInfo()`** to display customer information

e. Override **`calculateEarnings()`** to return **`loyaltyPoints * 0.5`**

f. Create getters and setters for `deliveryAddress` and `loyaltyPoints`

g. Overload **`operator+=`** to add loyalty points

Class DeliveryDriver (Inheritance)

a. Create class **`DeliveryDriver`** as subclass of **`User`**

b. Declare **`private`** instance

variables: **`vehicleType`** (string), **`completedDeliveries`** (int), **`totalEarnings`** (double)

c. Create default **`constructor`**

d. Create parameterized **`constructor`**

e. Override **`displayInfo()`** to display driver information including average earnings per delivery

f. Override **`calculateEarnings()`** to return **`totalEarnings`**

g. Create `method` **`completeDelivery(double orderValue)`** that adds 15% of **`orderValue`** to **`totalEarnings.`**

h. Overload **`prefix operator++`** to increment `completedDeliveries`

i. Overload **`postfix operator++`** to increment `completedDeliveries`

j. Create getters for `vehicleType`, `completedDeliveries`, `totalEarnings`

Class Order (Aggregation + Composition)

a. Declare **`private`** instance variables:

- **`orderId`** (**`string`**)
- **`customer`** **(`Customer*`)** - aggregation
- **`driver`** **(`DeliveryDriver*`)** - aggregation
- **`items`** (**`FoodItem`***) - **dynamic array** (composition)
- **`itemCount`** **(`int`)**
- **`capacity`** **(`int`)**
- **`status`** (**`OrderStatus`**)
- **`totalOrders`** (**`static int`**)

b. Create default **`constructor`**

c. Create parameterized **`constructor`** accepting **`orderId`** and Customer pointer

d. Create **copy `constructor`** (Deep Copy)

e. Create **`destructor`**

f. Create method **`addItem(const FoodItem& item)`** that adds items with dynamic resizing

g. Create method **`assignDriver(DeliveryDriver* drv)`**

h. Create method **`updateStatus(OrderStatus newStatus)`** that:

- Updates `status`
- If **`DELIVERED`**: calls driver's **`completeDelivery()`**, increments driver's counter, adds loyalty points to customer

i. Create method **`calculateTotal()`** that returns sum of all items

j. Create method **`displayOrder()`** that displays complete order details

k. Create static function **`getTotalOrders()`**

l. Create getters for `orderId`, `customer`, `driver`, `status`, `itemCount`

m. Overload **`operator+=`** to add `FoodItem`

n. Overload **`operator+`** to combine two orders into new order

o. Overload **`operator<<`** `(friend)` to print order details

p. Overload **`operator>`** `(friend)` to compare orders with total price

q. Overload **`operator[]`** to access items by index (both const and non-const versions)

File Operations

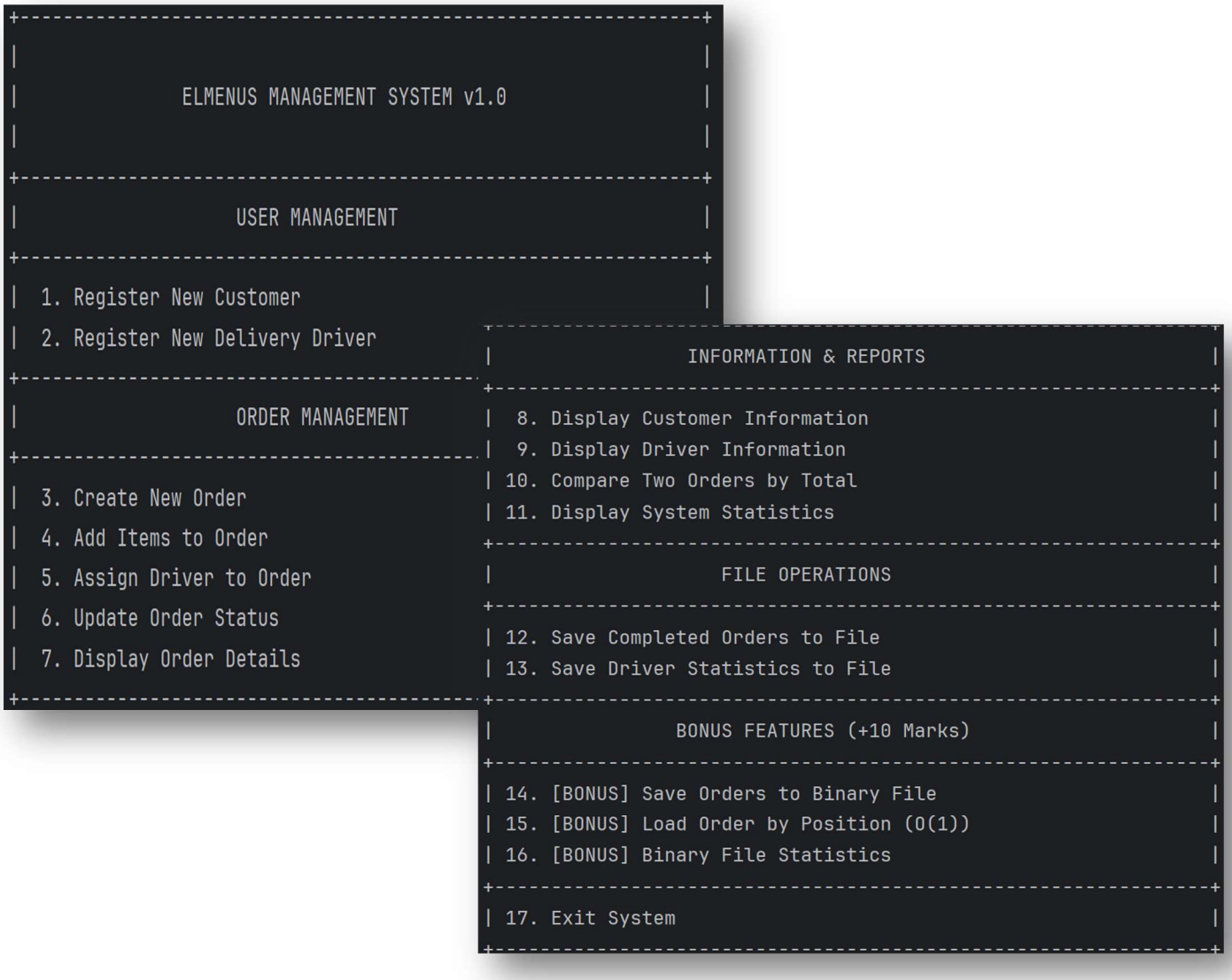
a. Create function that saves all completed orders to completed_orders.txt with order details

b. Create function that saves driver statistics to driver_stats.txt with earnings and delivery counts

c. Both functions should write to both screen and file

In Main.cpp

Implement a menu-driven program with the following options:



- Loop through menu until exit
- Validate all inputs
- Handle errors appropriately
- Free all dynamic memory before exit

BONUS: Binary File Direct Access (10 Bonus Marks)

a. Create a fixed-size struct OrderRecord with all order data

b. Implement function to save orders to binary file orders.dat

c. Implement function to load order by position in **O(1) time** using **seekg**() and direct byte offset calculation

d. Add menu options for binary file operations

e. Display time taken to retrieve order proving O(1) access

Writing Good Quality Code

I. Follow C++ coding style: http://geosoft.no/development/cppstyle.html

2. Use meaningful variable names

3. Add comments for complex sections

Grading Rubric (Total: 100 Marks)

Component	Marks
Inheritance & Polymorphism & Abstraction	25
Operator Overloading with Chaining (+=, +, <<, >, [], ++)	10
Dynamic Memory Management	5
Aggregation & Composition	25
File Operations (save orders and driver stats)	10
Enums & Static Members	5
Menu Implementation & Functionality	10
Code Quality (style, naming, comments, headers)	10
BONUS: Binary Direct Access	10

Submission Checklist

- *All .cpp and .h files with proper headers*
- *Proper folder naming: Assign2_ID1_ID2*
- *Compressed as .zip file*
- *All classes implemented correctly*
- *Operators overloaded properly*
- *Deep copy in copy constructor*
- *No memory leaks*
- *Clean code following C++ standards*
- *File operations working*
- *Menu fully functional*