**Assignment Number 2**

**CL-1004 Object Oriented Programming**

**Spring 2025**

**Instructions:**

* Partially or fully **copied assignments** will be marked as **zero**.
* Submission will be **Online** on Google Classroom with **Screenshots** of running code in a word file.
* Also Submit .cpp files separately of each question.
* Late submissions are not allowed.
* Solve all the questions in the given order.
* Write the programs with clarity and add comments where necessary.

**Carefully reallocate memory to store plural words.**

* + You cannot change the function prototypes given in the questions.
  + You cannot use break or goto statements. Breaks are allowed in switch cases.
  + Built-in string functions are not allowed.
  + Do not use new/extra strings wherever mentioned in the questions.
  + DO NOT USE ANY LIBRARIES LIKE VECTORS ETC.

**Q1: Turn-Based Strategy Game Assignment Breakdown . (50 marks)**

YOU MUST DEALLOCATE ALL MEMORY PROPERLY, YOUR CODE SHOULD NOT HAVE ANY **MEMORY LEAKS** OR **DANGLING POINTERS**.

#### **Part 1: Unit Structure (10 Marks)**

* **Task**: Implement the Unit structure that holds unit information such as unitType, health, attack, defense, movementRange, and position (nested structure for x and y coordinates).
* **Functionality**:
  + Create a constructor to initialize the values of the unit.
  + Implement methods:
    - moveUnit(int newX, int newY) – Move the unit on the map.
    - isAlive() – Returns true if the unit's health is above zero.
    - attackUnit(Unit\* enemyUnit) – Handles combat logic between two units.
    - displayInfo() – Displays the unit's current status (health, attack, position).

#### **Part 2: Map Structure (8 Marks)**

* **Task**: Implement the Map structure, which represents a 2D grid of units.
* **Functionality**:
  + grid (2D array of pointers to Unit objects) – The grid where units are placed.
  + Methods:
    - initializeMap() – Initializes the grid with empty cells.
    - addUnitToMap(Unit\* unit, int x, int y) – Adds a unit at the specified position.
    - removeUnitFromMap(int x, int y) – Removes the unit from the specified position.
    - displayMap() – Displays the map with positions of all units.

#### **Part 3: Player Structure (7 Marks)**

* **Task**: Implement the Player structure, which holds the player's name and units.
* **Functionality**:
  + Methods:
    - addUnit(Unit\* newUnit) – Adds a unit to the player’s list of units.
    - removeUnit(Unit\* unit) – Removes a unit from the player’s list of units.
    - displayUnits() – Displays the details of all units controlled by the player.

#### **Part 4: Game Structure (10 Marks)**

* **Task**: Implement the Game structure, which controls the game logic and players' turns.
* **Functionality**:
  + nextTurn() – Advances to the next player's turn.
  + performAction(int x, int y, int newX, int newY) – Executes a movement action for the current player.
  + checkVictory() – Checks if any player has won the game.
  + displayGameState() – Displays the current game state, including the map and whose turn it is.
  + Constructor to initialize the game with two players, and destructor to clean up memory.

#### **Part 5: Combat System (5 Marks)**

* **Task**: Implement the combat logic between two units.
* **Functionality**:
  + resolveCombat(Unit\* attacker, Unit\* defender) – Calculates the outcome of a combat between two units (damage calculation).
  + Ensure that health is updated and units are removed if their health drops below zero.

#### **Part 6: Main Function and Game Simulation (5 Marks)**

* **Task**: Implement the main function to initialize the players, units, map, and game, then simulate gameplay.
* **Functionality**:
  + Create two players, each controlling multiple units.
  + Place units on the map and perform sample actions (movement, attack).
  + Implement a game loop to simulate turns and combat between units.
  + Display the final result when a player wins.

#### **Part 7: Memory Management and Cleanup (5 Marks)**

* **Task**: Ensure proper memory management throughout the program.
* **Functionality**:
  + Use dynamic memory allocation (DMA) to allocate memory for units, players, and the map.
  + Properly deallocate memory to avoid memory leaks, especially in the Game destructor and after actions like removing units from the map.

**Question 2: Fantasy Adventure Game (50 marks)**

You are designing a **Fantasy Adventure Game** where players explore different locations, encounter enemies, and collect loot. The game involves a dynamic world consisting of multiple areas with different types of terrain. Players can interact with objects, gain experience, and increase their stats.

In this game, **enums** are used for different types of items, enemies, and terrains, while **multi-level nested structures** represent complex data such as **character stats**, **inventory**, and **areas**.

You will implement the following structures and functionalities for the game. The task is to build **multi-level nested structures**, **implement enums**, and manage the interaction between various entities in the game world.

#### **Part 1: Define Enums (5 Marks)**

**ItemType Enum**: Define an enum ItemType that represents different types of items:

* 1. WEAPON
  2. ARMOR
  3. POTION
  4. MISC

**EnemyType Enum**: Define an enum EnemyType that represents different types of enemies:

* 1. GOBLIN
  2. TROLL
  3. DRAGON
  4. SKELETON

**TerrainType Enum**: Define an enum TerrainType that represents different types of terrain:

* 1. FOREST
  2. CAVE
  3. MOUNTAIN
  4. VILLAGE

#### **Part 2: Define Multi-Level Nested Structures (15 Marks)**

**Item Structure** (5 Marks):

* 1. Define a structure Item that holds information about an item in the game. Each item has:
     1. name (string)
     2. type (ItemType enum)
     3. value (integer, monetary value)
     4. weight (double, weight in kg)
  2. Implement a function describeItem() that provides a description of the item.

**Enemy Structure** (5 Marks):

* 1. Define a structure Enemy that represents an enemy in the game. Each enemy has:
     1. name (string)
     2. type (EnemyType enum)
     3. health (integer)
     4. attackPower (integer)
     5. defensePower (integer)
  2. Implement a function attackEnemy() that simulates an attack on the enemy, decreasing their health based on attack power.

**Character Structure** (5 Marks):

* 1. Define a structure Character that represents the player character. Each character has:
     1. name (string)
     2. level (integer)
     3. experience (integer)
     4. health (integer)
     5. attackPower (integer)
     6. defensePower (integer)
     7. inventory (array of Item pointers)
     8. inventorySize (integer)
  2. Implement methods:
     1. gainExperience(int exp) – Increases experience and levels up the character if experience exceeds a threshold.
     2. useItem(Item\* item) – Uses an item from the character's inventory and applies its effect.
     3. displayCharacterInfo() – Displays the character's stats, including inventory.

#### **Part 3: Define the World Structure (15 Marks)**

**Terrain Structure** (5 Marks):

* 1. Define a structure Terrain that represents a terrain in the game world. Each terrain has:
     1. name (string)
     2. type (TerrainType enum)
     3. enemies (array of pointers to Enemy objects)
     4. numEnemies (integer)
     5. items (array of pointers to Item objects)
     6. numItems (integer)
  2. Implement methods:
     1. addEnemy(Enemy\* enemy) – Adds an enemy to the terrain.
     2. addItem(Item\* item) – Adds an item to the terrain.
     3. displayTerrainInfo() – Displays the terrain's information, including the enemies and items present.

**Area Structure** (5 Marks):

* 1. Define a structure Area that represents an area on the map. Each area contains:
     1. name (string)
     2. terrain (pointer to Terrain structure)
     3. connectedAreas[] (array of pointers to Area structures, for representing interconnected areas)
     4. numConnectedAreas (integer)
  2. Implement methods:
     1. addConnectedArea(Area\* area) – Adds a connected area to the current area.
     2. displayAreaInfo() – Displays information about the area, including the terrain and connected areas.

**Game World Structure** (5 Marks):

* 1. Define a structure GameWorld that represents the entire game world. The game world consists of:
     1. areas[] (array of pointers to Area structures)
     2. numAreas (integer)
  2. Implement methods:
     1. addArea(Area\* area) – Adds an area to the world.
     2. displayWorldInfo() – Displays information about the game world, including all areas and terrains.

#### **Part 4: Main Game Simulation (10 Marks)**

1. **Simulation**:
   1. In the main function, simulate the following:
      1. Create a few Items, Enemies, and Terrains.
      2. Create some Characters and place them in a GameWorld with connected areas.
      3. Simulate moving a character through different areas, interacting with terrain (e.g., fighting enemies, collecting items).
      4. Display the status of the character after each action (e.g., after attacking an enemy or using an item).
   2. Handle the **character leveling up** and **item usage** correctly. Ensure that the character's stats are updated appropriately after using items or gaining experience.

**Question 3: Multi-Tiered Resource and Asset Management System for Space Exploration (50 marks)**You are tasked with building a **Resource and Asset Management System** for a space exploration organization. The system must manage complex relationships between **resources**, **crew members**, **missions**, and **vehicles**. These entities interact with each other, and resources such as fuel, food, and medical supplies must be managed effectively to ensure mission success.

The system will incorporate **nested enums** within classes, which will represent various types of resources, missions, and vehicles in the space exploration context.

#### **Part 1: Define Nested Enums (10 Marks)**

**ResourceType Enum (Nested inside** ResourceManager **class)**: Define an enum ResourceType to represent different types of resources.

* 1. FUEL
  2. FOOD
  3. WATER
  4. MEDICAL\_SUPPLIES
  5. TECHNOLOGY
  6. MATERIALS

**MissionType Enum (Nested inside** Mission **class)**: Define an enum MissionType to represent different types of missions:

* 1. EXPLORATION
  2. TRANSPORT
  3. DEFENSE
  4. REPAIR
  5. COLONIZATION

**VehicleType Enum (Nested inside** Vehicle **class)**: Define an enum VehicleType to represent different types of vehicles:

* 1. SHUTTLE
  2. ROVER
  3. SPACE\_STATION
  4. FREIGHTER
  5. PROBE

#### **Part 2: Define Multi-Level Nested Structures (20 Marks)**

**Resource Structure (Nested inside** ResourceManager **class)** (5 Marks):

* 1. Define a structure Resource to represent a resource in the system:
     1. type (ResourceType enum)
     2. quantity (double)
     3. value (double, monetary value)
     4. expirationDate (string, in format "DD/MM/YYYY")
     5. **Method**: consume(double amount) – Reduces the resource quantity by the specified amount.

**Crew Member Structure (Nested inside** CrewManager **class)** (5 Marks):

* 1. Define a structure CrewMember to represent crew members:
     1. name (string)
     2. role (string)
     3. rank (string)
     4. experienceYears (integer)
     5. assignedMissions[] (array of pointers to Mission objects)

**Methods**:

* + - 1. assignMission(Mission\* mission) – Assigns a mission to the crew member.
      2. displayInfo() – Displays detailed information about the crew member.

**Vehicle Structure (Nested inside** VehicleManager **class)** (5 Marks):

* 1. Define a structure Vehicle to represent a vehicle in the system:
     1. name (string)
     2. type (VehicleType enum)
     3. fuelLevel (double)
     4. crewCapacity (integer)
     5. currentCrew[] (array of pointers to CrewMember objects)

**Methods**:

* + - 1. refuel(double amount) – Refuels the vehicle.
      2. assignCrew(CrewMember\* member) – Assigns a crew member to the vehicle.

**Mission Structure (Nested inside** MissionManager **class)** (5 Marks):

* 1. Define a structure Mission to represent a mission:
     1. name (string)
     2. type (MissionType enum)
     3. startDate (string)
     4. endDate (string)
     5. status (string, e.g., "Planned", "In Progress", "Completed")
     6. assignedCrew[] (array of pointers to CrewMember objects)
     7. assignedVehicle (pointer to Vehicle object)

**Methods**:

* + - 1. assignCrewMember(CrewMember\* member) – Assigns a crew member to the mission.
      2. updateMissionStatus(string status) – Updates the mission status (e.g., "In Progress", "Completed").
      3. displayMissionDetails() – Displays mission details.

#### **Part 3: Management System Logic (15 Marks)**

**Resource Management System** (5 Marks):

* 1. Define a class ResourceManager that manages resources:
     1. resources[] (array of pointers to Resource objects)
     2. **Methods**:
        1. addResource(Resource\* resource) – Adds a new resource.
        2. consumeResource(string resourceName, double amount) – Consumes a specified amount of a resource.
        3. replenishResource(string resourceName, double amount) – Replenishes a specified resource.
        4. displayResources() – Displays the status of all resources in the system.

**Crew Management System** (5 Marks):

* 1. Define a class CrewManager that manages crew members:
     1. crew[] (array of pointers to CrewMember objects)

**Methods**:

* + - 1. addCrewMember(CrewMember\* member) – Adds a new crew member.
      2. assignCrewToMission(string crewName, Mission\* mission) – Assigns a crew member to a mission.
      3. displayCrewInfo() – Displays detailed information about all crew members.

**Vehicle Management System** (5 Marks):

* 1. Define a class VehicleManager that manages vehicles:
     1. vehicles[] (array of pointers to Vehicle objects)

**Methods**:

* + - 1. addVehicle(Vehicle\* vehicle) – Adds a new vehicle.
      2. assignCrewToVehicle(string vehicleName, CrewMember\* member) – Assigns a crew member to a vehicle.
      3. refuelVehicle(string vehicleName, double amount) – Refuels a vehicle.
      4. displayVehicleInfo() – Displays detailed information about all vehicles.

#### **Part 4: Main Game Logic and Simulation (10 Marks)**

1. **Simulation** (10 Marks):
   1. In the main function, simulate the following actions:
      1. Create and add resources (e.g., fuel, food, materials) to the ResourceManager.
      2. Create and add crew members to the CrewManager.
      3. Create and add vehicles (e.g., shuttle, rover) to the VehicleManager.
      4. Create and assign missions (e.g., exploration, transport) to crew members and vehicles.
      5. Simulate mission progress, including:
         1. Resource consumption (e.g., fuel, food).
         2. Crew members working on missions.
         3. Vehicles being used for transport or exploration.
      6. Display the status of resources, crew, and vehicles after each action.

**Question 4: Online Survey System with Role-Based Access (50 marks)**

You are tasked with building an **Online Survey System** where users can create, distribute, and fill out surveys. The system will have **two roles**: **Admin** and **User**. The **Admin** will be responsible for creating surveys, managing questions, viewing responses, and generating analytics. The **User** can only fill out surveys and submit their responses. The system should be able to track responses and generate reports for analysis.

#### **Roles and Responsibilities**:

**Admin**:

* 1. Create and manage surveys.
  2. Add and manage different types of questions in the survey.
  3. View survey results and analytics, such as response distribution and statistical data.

**User**:

* 1. Fill out surveys by answering the questions.
  2. Submit responses to surveys created by the Admin.

#### **Key Requirements**:

**Survey Creation**:

* 1. Admin should be able to create surveys with multiple types of questions, including multiple choice, text input, and rating scale questions.
  2. Admin should be able to manage (add, edit, remove) questions for a survey.

**Survey Responses**:

* 1. Users should be able to fill out surveys, providing answers to the questions.
  2. Responses should be stored, and users should only be allowed to fill out the survey once.

**Analytics and Reporting**:

* 1. Admin should be able to generate reports for each survey that include:
     1. The number of responses for each question.
     2. The percentage breakdown of answers for multiple choice and rating scale questions.
     3. Graphical representation of survey data (e.g., bar charts, pie charts).

**Role-Based Access**:

* 1. **Admin** has access to all survey management and reporting features.
  2. **User** can only access and submit responses to surveys, without access to survey management or analytics.

**Dynamic Memory Allocation**:

* 1. Use dynamic memory allocation to manage the creation of surveys, questions, and responses efficiently.

**Classes and Objects**:

* 1. Implement the system using classes to represent users, surveys, questions, answers, and reports.

**Enums**:

* 1. Define enums to represent question types (e.g., multiple choice, text input) and response types (e.g., yes/no, scale ratings).

### **Expected Outcomes**:

1. The system must be able to handle user authentication, including distinguishing between Admin and User roles.
2. Admins must be able to create surveys, add different types of questions, and view analytics.
3. Users must be able to fill out surveys and submit their responses, which are then tracked and stored in the system.
4. Admins should be able to generate reports and view responses to each question in the survey.
5. The system should properly manage memory, particularly for dynamically created surveys, questions, and responses.

This project is designed to test your understanding of **nested enums**, **multi-level nested structures**, **dynamic memory allocation**, **classes with constructors and destructors**, and **role-based access control** in a real-world scenario.