Department of Computer and Software Engineering- ITU
CE101L: Object Oriented Programming Lab

Course Instructor: Nadir Abbas	**Dated: 28/01/2025**
Teaching Assistant: Zainab, Nahal, Bilal	**Semester: Spring 2025**
Session: 2024-2028	**Batch: BSCE24**

Lab 2. Implementation of Structures and Nested Structures

Name	Roll number	Obtained Marks/35

Checked on: ______________________________

Signature: ______________________________

Objective

The objective of this lab is to help students understand and apply concept of structs and nested structs.

Equipment and Component

Component Description	Value	Quantity
Computer	Available in lab	1

Conduct of Lab

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

Theory and Background

Struct and Nested Struct:

A struct in C++ is a user-defined data type that allows grouping different types of variables, called members, under one name. A nested struct is a struct defined inside another struct. It allows creating complex data structures with hierarchical relationships. For example:

```
struct Address {
    string street;
    string city;
};

struct Person {
    string name;
    Address address;
};
```

Constructors:

Constructors are special member functions that are automatically called when an object of a class or struct is created. They initialize objects to a valid state and can be overloaded to accept different types or numbers of arguments. A constructor does not return a value. Example:

```
struct Person {
    string name;
    int age;
    Person(string n, int a) : name(n), age(a) {}  // Constructor
};
```

Initialization List:

An initialization list in C++ allows initializing member variables directly at the time of object creation, before the constructor body executes. It's efficient as it avoids default initialization and immediate reassignment. Example:

```
struct Person {
    string name;
    int age;
    Person(string n, int a) : name(n), age(a) {}  // Initialization list
};
```

Tasks

Task 1: Accept the assignment posted in Google Classroom and after accepting clone the repository to your computer for this ensure you have logged into github app with your account.

Task 2: Solve the given problems written after task instructions, write code through IDE like CLion

Task 3: Ensure your code/solution is in the cloned folder.

Task 4: Commit and Push the changes through the Github App

Write code in functions, in files **functions.cpp**, **functions.h** and **main.cpp** after completing each part, verify through running code using **"make run"** on Cygwin.

Q1: Define Structures

1. **Project:**
    - Members:
        - projectId (integer)
        - projectName (string)
        - status (bool)
    - Constructors:
        - **Default Constructor**: Initializes projectId to 0, projectName to an empty string, and status to false.
        - **Parameterized Constructor**:
          Project (int id, string name, bool status)
          Accepts values for all fields and initializes them using an initialization list.
2. **Performance:**
    - Members:
        - rating (float)
        - feedback (string)
    - Constructors:
        - **Default Constructor**: Initializes rating to 0 and feedback to an empty string.
        - **Parameterized Constructor**:
          Performance (float rating, string feedback)
          Accepts values for all fields.
3. **Employee:**
    - Members:
        - empId (integer)
        - name (string)
        - department (string)
        - projects (*Project array of project struct)
        - numProjects (int)
        - performance (Performance)
    - Constructors:
        - **Default Constructor**: Initializes empId to 0 name, department to an empty string projects to

nullptr, numProjects to 0 performance initialized using the Performance structure's default constructor.

        - **Parameterized Constructor**:
          Employee (int id, string name, string department, Performance performance)
          Accepts id, name, department, and performance as input parameters.
            - Initializes the respective fields using an **initialization list**.
            - Sets projects to nullptr and numProjects to 0 as no projects are assigned initially.

4. **Company:**

    - Members:
        - name (string)
        - numEmployees (int)
        - employees (**Employee* array of employees)**
    - Constructors:
        - **Default Constructor**: Initializes name to empty string numEmployees to 0 employees to nullptr
        - **Parameterized Constructor**:
          Company (string name)
          Accepts the name of the company as an input parameter.
            - Initializes name using an **initialization list**.
            - Sets numEmployees to 0 and employees to nullptr by default.

Q2: Assign Project to Employee

Implement the following function outside the structures:

void assignProject(Employee& employee, const Project& project)

Task:

- Accept an Employee object by reference and a Project object by constant reference.
- Dynamically add the Project to the employee's projects array after regrowing it.
- Increment the numProjects field.

Q3: Evaluate Employee Performance

Implement the following function outside the structures:

void evaluatePerformance(Employee& employee, Performance& performance)

Task:

- Accept an Employee object by reference and a Performance object by reference.
- Update the employee's performance field with the new evaluation.

Q4: Retrieve Employees by Department

Implement the following function outside the structures:

Employee* getEmployeesByDepartment(Company &company, int& count, string& department)

Task:

- Accept a Company object, a department name, and an integer to hold the count of employees.
- Filter employees in the given department and return a pointer to a dynamically created array containing these employees.
- Update the count with the number of employees in the department.

Q5: List All Projects of an Employee

Implement the following function:

Project* listEmployeeProjects(Employee& employee)

Task:

- Accept an Employee object by constant reference.
- Return all the active project list or nullptr if no active projects.

Q6: Find the Top-Performing Employee

Implement the following function:

Employee findTopPerformer(const Company& company)

Task:

- Accept a Company object.
- Iterate through all employees and return the one with the highest performance rating. if more than one return first employee with highest ratting

Q7: Count Active Projects in the Company

Implement the following function:

int countActiveProjects(const Company& company)

Task:

- Accept a Company object.
- Count and return the total number of active projects across all employees.

Q8: Remove an Employee from a Company

Implement the following function:

void removeEmployee(Company& company, int empId)

Task:

- Accept a Company object and an employee ID (empId).
- Remove the employee with the given ID from the company’s employees array.
- Update the numEmployees field.

Q9: Search an employee in the company

Implement the following function:

bool searchEmployee(Company& company, int empId)

Task:

- Accept a Company object and an employee ID (empId).
- search the employee with the id and return 0 if not present and 1 if present

Q10: Add employee in company

Implement the following function:

void addEmployee(Company& company, Employee& employee)

Task:

- Add the given employee in the company employee array after regrowing it

Tasks

Task 1: Accept the assignment posted in Google Classroom and after accepting clone the repository to your computer for this ensure you have logged into github app with your account.

Task 2: Solve the given problems written after task instructions, write code through IDE like CLion

Task 3: Ensure your code/solution is in the cloned folder.

Task 4: Commit and Push the changes through the Github App

Write code in functions, in files **functions.cpp**, **functions.h** and **main.cpp** after completing each part, verify through running code using **“make run”** on Cygwin.

Assessment Rubric for Lab

Performance metric	CLO	Able to complete the task over 80% (4-5)	Able to complete the task 50-80% (2-3)	Able to complete the task below 50% (0-1)	Marks
1. Realization of experiment	1	Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed.	Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed.	Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed.	
2. Conducting experiment	1	Able to make changes and answered all questions.	Partially able to make changes and few incorrect answers.	Unable to make changes and answer all questions.	
3. Computer use	2	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	3	Actively engages and cooperates with other group member(s) in effective manner.	Cooperates with other group member(s) in a reasonable manner but conduct can be improved.	Distracts or discourages other group members from conducting the experiment	
5. Laboratory safety and disciplinary rules	3	Code comments are added and does help the reader to understand the code.	Code comments are added and does not help the reader to understand the code.	Code comments are not added.	
6. Data collection	3	Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap.	Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables.	Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy.	
7. Data analysis	4	Solution is efficient, easy to understand, and maintain.	A logical solution that is easy to follow but it is not the most efficient.	A difficult and inefficient solution.	
Total (out of 35):					