Department of Computer and Software Engineering – ITU

CE101T: Object Oriented Programming

Course Instructor: Nadir Abbas	**Dated: 09/02/2025**
Teaching Assistant: Zainab, Nahal, Bilal	**Semester: Spring 2025**
Session: 2024-2028	**Batch: BSCE2024**

Assignment 3. Advanced Structs, Nested Structures, Constructors, and JSON File Handling

Name	Roll number	Obtained Marks/35

Checked on: _____

Signature: _____

Submission:

- Email instructor or TA if there are any questions. You cannot look at others' solutions or use others' solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.
- Submission after due time will not be accepted.

In this assignment you have to do following tasks:

Task 1: Ensure that you have installed all three softwares in your personal computer (Github, Cygwin & CLion). Now, accept the assignment posted in classroom (e.g Google, LMS etc) and after accepting, clone the repository to your computer. Make sure you have logged into the github app with your account.

Task 2: Open Cygwin app, Move to your code directory with following command "cd <path_of_folder>"
<path_of_folder> can be automatically populated by dragging the folder and dropping it to the cygwin window.
Run the code through Cygwin, use command "make run", to get the output of the code

Task 3: Solve the given problems, write code using **CLion** or any other IDE.
Task 4: Keep your code in the respective git cloned folder.
Task 5: Commit and Push the changes through the Github App
Task 6: Write the code in separate files **(functions.h and functions.cpp**). Ensure that file names are in lowercase (e,g **main.cpp**).

Task 7: Run '**make run**' to run C++ code
Task 8: Run '**make test**' to test the C++ code

Problem Statement:
The goal is to implement a University Management System that supports nested structures, dynamic resizing of arrays, JSON file handling, and separate functions for performing key operations. The assignment is divided into multiple tasks for better understanding.

Q1: Define Structures

1. **Address**

   Members:
   - street (string)
   - city (string)
   - zipCode (string)

   Constructors:
   - **Default Constructor**:
     Initializes street, city, and zipCode to empty strings.
     Address();
   - **Parameterized Constructor**:
     Accepts values for all fields and initializes them using an initialization list.
     Address(string street, string city, string zipCode);

2. **Teacher**

   Members:
   - teacherId (int)
   - firstName (string)
   - lastName (string)
   - address (Address)

   Constructors:
   - **Default Constructor**:
     Initializes teacherId to 0, firstName and lastName to empty strings, and address using the default constructor of the Address structure.
     Teacher();
   - **Parameterized Constructor**:
     Accepts values for teacherId, firstName, lastName, and address and initializes them using an initialization list.
     Teacher(int teacherId, string firstName, string lastName, Address address);

3. **Course**

   Members:
   - courseCode (string)
   - courseName (string)
   - credits (int)
   - teacher (Teacher)

   Constructors:

- **Default Constructor**:
  Initializes courseCode and courseName to empty strings, credits to 0, and teacher using the default constructor of the Teacher structure.
  Course();
- **Parameterized Constructor**:
  Accepts values for courseCode, courseName, credits, and teacher and initializes them using an initialization list.
  Course(string courseCode, string courseName, int credits, Teacher teacher);

4. **Grade Report**

Members:

- courseCode (string)
- grade (float)

Constructors:

- **Default Constructor**:
  Initializes courseCode to an empty string and grade to 0.
  GradeReport();
- **Parameterized Constructor**:
  Accepts values for courseCode and grade and initializes them using an initialization list.
  GradeReport(string courseCode, float grade);

5. **Student**

Members:

- studentId (int)
- firstName (string)
- lastName (string)
- address (Address)
- courses (Course* array of Course structures)
- gradeReport (GradeReport* array of Grade Report structures)
- numCourses (int)

Constructors:

- **Default Constructor**:
  Initializes studentId to 0, firstName and lastName to empty strings, address using the Address structure's default constructor, courses and gradeReport to nullptr, and numCourses to 0.
  Student();
- **Parameterized Constructor**:
  Accepts values for studentId, firstName, lastName, and address.
  Initializes courses and gradeReport to nullptr and numCourses to 0.
  Student(int studentId, string firstName, string lastName, Address address);

6. **Classroom**

Members:

- classroomId (int)
- students (Student* array of Student structures)
- numStudents (int)

Constructors:

- **Default Constructor**:
  Initializes classroomId to 0, students to nullptr, and numStudents to 0**.**
  Classroom();
- **Parameterized Constructor**:
  Accepts classroomId and initializes students to nullptr and numStudents to 0.
  Classroom(int classroomId);

7. **University**

Members:

- name (string)
- classrooms (Classroom* array of Classroom structures)
- numClassrooms (int)

Constructors:

- **Default Constructor**:
  Initializes name to an empty string, classrooms to nullptr, and numClassrooms to 0**.**
  University();
- **Parameterized Constructor**:
  Accepts name as an input parameter and initializes it using an initialization list.
  Sets classrooms to nullptr and numClassrooms to 0.
  University(string name);

Q2: void addNewUniversity(University& university, const string& filename);

Task:

- Accept a University object by reference and a filename as a string.
- Read the university.json file into a JSON array.
- Search the array for a university with the same name. If found, replace it; otherwise, append the new university object.
- Serialize the updated array and write it back to the file.

Q3: void addClassroomToUniversity(University& university, const Classroom& classroom);

Task:

- Accept a University object by reference and a Classroom object by constant reference.
- Dynamically resize the classrooms array in the University structure.
- Append the new Classroom object to the array.

- Increment the numClassrooms field in the University structure.

Q4: void addStudentToClassroom(Classroom& classroom, const Student& student);

Task:

- Accept a Classroom object by reference and a Student object by constant reference.
- Dynamically resize the students array in the Classroom structure.
- Append the new Student object to the array.
- Increment the numStudents field in the Classroom structure.

Q5: void addCourseToStudent(Student& student, const Course& course);

Task:

- Accept a Student object by reference and a Course object by constant reference.
- Dynamically resize the courses array in the Student structure.
- Append the new Course object to the array.
- Dynamically resize the gradeReport array and add a new GradeReport object with an empty grade.
- Increment the numCourses field in the Student structure.

Q6: void addGrade(Student& student, const string& code, const float grade);

Task:

- Accept a Student object by reference, a course code as a string, and a grade as a float.
- Search the gradeReport array for the specified course code.
- Update the grade for the corresponding course.

Q7: Student* highestAchiever(Classroom& classroom);

Task:

- Accept a Classroom object by reference.
- Calculate the average grade for each student using their gradeReport.
- Return the student with the highest average grade.

Q8: Student* mostCourses(Classroom& classroom);

Task:

- Accept a Classroom object by reference.
- Iterate through the students array to find the student with the highest numCourses.
- Return the student object.

Q9: float universityAverageGrade(const University& university);

Task:

- Accept a University object by constant reference.
- Calculate the average grade of all students in all classrooms across the university.
- Return the overall average grade.

Q10: void displayUniversityDetails(const University& university);

Task:

- Accept a University object by constant reference.
- Display the university name, number of classrooms, and details of each classroom, including students, courses, and grades.

Q11: void removeStudentFromClassroom(Classroom& classroom, int studentId);

Task:

- Accept a Classroom object by reference and a studentId as an integer.
- Traverse the students array to find the student with the matching studentId.
- Remove the student by shifting subsequent elements of the array.
- Decrement the numStudents field in the Classroom structure.

Q12: void removeClassroomFromUniversity(University& university, int classroomId);

Task:

- Accept a University object by reference and a classroomId as an integer.
- Traverse the classrooms array to find the classroom with the matching classroomId.
- Remove the classroom by shifting subsequent elements of the array.
- Decrement the numClassrooms field in the University structure.

Q13: void removeCourseFromStudent(Student& student, const string& courseCode);

Task:

- Accept a Student object by reference and a course code as a string.
- Traverse the courses array to find the course with the matching courseCode.
- Remove the course by shifting subsequent elements of the array.
- Remove the corresponding entry in the gradeReport array.
- Decrement the numCourses field in the Student structure.

Q14: void removeUniversityFromFile(const string& name, const string& filename);

Task:

- Accept the university name as a string and a filename as a string.
- Read the university.json file into a JSON array.
- Traverse the array to find the university object with the matching name.
- Remove the university object from the array.
- Write the updated array back to the file.

Note:
Json file structure:

```
[
  {
    "name": "Tech University",
    "numClassrooms": 1
  },
  {
    "name": "Arts University",
    "numClassrooms": 2
  }
]
```

Assessment Rubric for Assignment

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	3	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	2	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	4	Document submission timely.	Document submission late.	Document submission not done.	
4. Teamwork	4	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	2	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	2	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	3	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):					