ITM
IT MUSTAKBIL

Final Examination (Morning Batch)

Full Name:

Roll No: **Section:**

Object Oriented Programming **3 hours**

Problem Solving & Practical Programming Semester 2

Spring (2025)

Instructions:

- You must answer **all** questions. There are **no optional** questions.
- You are responsible for ensuring your answers are clear and unambiguous.
- You must upload your answers in the quiz form uploaded on Google Classroom.
- Write your full name, roll number and section in the boxes at the top of the page.
- **You may use any offline notes, handouts, notes or resources except help taken from other students of ITM and their work, and except LLMs.**
- This exam affords **zero tolerance** to students found using dishonest or unfair means.

Information:

- The total marks for this paper are 45.
- The number of marks for each question or part question are shown in brackets [ ].
- Students may only be awarded whole number marks, with no partial marks.
- There are a total of 5 pages in this paper.
- There are **three sections** in this paper; A) Simple Programs, B) Medium Programs and C) Complex Programs.

C++ Programming Marks
/ 45

systems

Section A – 40 Minutes

Simple Programs

Examiner Use

1 You are tasked with writing a **C++ program** for a company that processes customer orders using an **object-oriented approach**. Each customer has details such as **name**, **address**, **email**, and **password**. After signing in, a customer can place an order that includes multiple items. The system should use **two classes**: one for the **customer** and another for the **order**.

The Customer class stores the customer's name and address and is associated with an **Order object**. The Order class includes attributes like the **orderDate**, **status**, and arrays to store **itemPrices** and **itemWeights** for up to 5 items. It also provides the following member functions:

- **addItem**(double price, double weight).
- **calcSubTotal()** – calculates subtotal of item prices.
- **calcTax()** – calculates 10% tax on subtotal.
- **calcTotal()** – calculates total amount including tax.
- **calcTotalWeight()** – calculates total weight of all items.

Use the following technical specifications outlined below to develop a program that captures customer and order information, performs required calculations, and displays a complete summary of the order.

Technical Specifications:

1. Use two classes: **Customer** and **Order.**
2. Use arrays or vectors to store up to 5 item prices and weights.
3. Add at least 2 items using **addItem()** in the **main()** function.
4. Associate the **Order** object with the **Customer** object.
5. Display customer info and order summary: **orderDate, status, subTotal, tax, totalAmount,** and **totalWeight.**
6. Use proper access specifiers (**private**, **public**).
7. Apply **dynamic memory allocation (DMA)** where appropriate.

Note: Correct use of access control and DMA (Dynamic Memory Allocation) will result in 3% bonus weightage in the final grade.

[10]

systems

Section B – 60 Minutes

Medium Programs

Examiner Use

2 You are tasked with writing a C++ program to model a **Teaching Assistant (TA)** in a university using an object-oriented approach. A TA acts both as a **student** and a **teacher**: they attend classes and study like a student but also assist in delivering lectures and managing labs like a teacher.

The program should use an **abstract base class Person** with attributes name, **ssn,** and **age,** and a pure virtual method **displayInfo()** that derived classes override. The **Student class** inherits from **Person** and adds a private attribute major, with a **study()** method. The **Teacher class** also inherits from Person and adds a private attribute subject, with a **teach()** method. The **TeachingAssistant** class inherits publicly from both Student and Teacher, adds an **assist()** method, and **overrides study()** and **teach()** to demonstrate polymorphism.

Use the following technical specifications outlined below to develop a program that creates a **TechingAssistant** object, display its information, and demonstrates its roles.

Technical Specifications:

1. Use an abstract base class Person with private attributes **name**, **ssn**, and **age.**
2. Declare a pure virtual method **displayInfo()** in Person.
3. Create **Student** and **Teacher** classes that inherit publicly from **Person**.
4. Student class adds private attribute major and public **method study().**
5. Teacher class adds private attribute subject and public **method teach().**
6. **TeachingAssistant** class inherits publicly from both Student and Teacher.
7. **TeachingAssistant** adds method **assist()** and overrides **study()** and **teach()**.
8. In **main(),** create a **TeachingAssistant** object, call **displayInfo(),** then call **study**(), **teach(), and assist().**
9. Use proper access specifiers (private, public).
10. Apply dynamic memory allocation (DMA) where appropriate.

Note:

Adding the following two methods to **TeachingAssistant** will lead to an extra **5% bonus** weightage:

- **research()** — simulates research work
- **grade()** — simulates grading assignments

[15]

systems

Section C – 80 Minutes

Complex Programs

Examiner Use

3 You are tasked with writing a C++ program for a **local crime reporting agency** that aims to improve operational efficiency by allowing users to report crimes, manage records, and view details. The system is **designed using object-oriented principles** including inheritance, polymorphism, encapsulation, and dynamic memory allocation.

The system consists of the following classes:

- **Location** – stores city and area information.
- **Evidence** – stores details of crime evidence.
- **Report** – a base class that stores reporter's name, CNIC, contact number (as private), and verification status. It includes functions to set and display report data.
- **CrimeReport** – inherits from Report and adds public attributes: crimeType and description. It overrides the base class methods to handle additional crime-specific information.

Use the following technical specifications outlined below to develop a program that accepts and displays crime reports using a menu-driven interface.

Technical Specifications:

1. Create a base **class Report** with private attributes: name, cnic, contactNumber, and a public attribute verificationStatus.
2. Create a derived class **CrimeReport** that inherits from Report and adds public attributes: crimeType and description.
3. Override Report's member functions in **CrimeReport** to include the extended details.
4. Use dynamic memory allocation to create an array of Report pointers based on user-defined size.
5. Implement a menu-driven interface with the following options:
    - Add a new crime report (input all relevant details).
    - Display all reports (including auto-generated Crime ID, current date/time using <ctime>, and verification status).
    - Exit the program.
6. Use the **rand() function** to generate a unique Crime ID for each report.
7. Use **ctime** to record and display the report submission time.
8. Use virtual functions in the base class to support runtime polymorphism.
9. Properly deallocate all dynamically allocated memory to avoid memory leaks.
10. Apply proper access specifiers (private, public) for data encapsulation.

Written Task:

Answering the following conceptual questions using block comments at the end of your code will lead to an extra 10% bonus weightage:

1. Identify and explain how inheritance and polymorphism are applied in the system.
2. Describe how the system encapsulates user details to protect sensitive information. Why is this important?
3. Discuss the purpose of dynamic memory allocation in this system. How is it implemented, and why is memory deallocation important?
4. Explain how the system uses timestamps to enhance the reporting functionality.
5. If the system were to be expanded to handle Accident Reports in addition to Crime Reports, suggest how the current design (using inheritance and polymorphism) would support this enhancement.
6. Propose one improvement to make the menu-driven interface more user-friendly. **[20]**

Blank Page

Rough Work / Additional Space