ITM
IT MUSTAKBIL

Final Examination (Evening 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 library that keeps track of members and the books they borrow using an **object-oriented approach**. Each member has details such as **name**, **membership ID**, **email**, and **contact number**. After registration, a member can borrow multiple books at a time. The system should use **two classes**: one for the **member** and another for the **borrowed books**.

The **Member class** stores the member's personal information and is associated with a **BorrowRecord** object. The **BorrowRecord** class includes attributes like **borrowDate**, **dueDays**, and arrays to store **bookTitles** and **bookPages** for up to 5 books. It also provides the following member functions:

- **addBook**(string title, int pages) – adds a book with its title and page count.
- **calcTotalPages()** – returns the total number of pages across all borrowed books.
- **calcDueAmount()** – calculates a due amount based on a fixed rate of Rs. 1 per page.
- **calcFine(int extraDays)** – calculates a fine of Rs. 10 per extra day after dueDay.
- **summary()** – displays full borrow record.

Use the following technical specifications outlined below to develop a program that handles member borrowing, calculates relevant stats, and displays a complete summary of borrowed books.

Technical Specifications:

1. Use two classes: **Member** and **BorrowRecord.**
2. Use arrays or vectors to store up to 5 book titles and page counts.
3. Add at least 2 books using **addBook()** in the **main()** function.
4. Associate the **BorrowRecord** object with the **Member object**.
5. Display member info and borrow summary: **borrowDate, dueDays, totalPages, dueAmount,** and sample fine for 3 extra days.
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]

Section B – 60 Minutes

Medium Programs

Examiner Use

2 You are tasked with writing a C++ program for a company that **manages employees** of different roles using an object-oriented approach. **Employees** can be **Managers**, **Developers**, or **Interns**. Each employee has attributes name and **employeeID**, and role-specific details.

The program should use an **abstract base class Employee** with private attributes name and **employeeID**, a pure virtual method **displayInfo(),** and a virtual method **isEqual()** to check equality based on employeeID. Manager, Developer, and Intern classes inherit publicly from Employee and add private role-specific attributes: department, **programmingLanguage**, and **durationMonths** respectively. Each derived class overrides **displayInfo().**

Use the following technical specifications outlined below to develop a program that creates employee objects, identifies their roles using runtime type checking, compares employees for equality, and displays their information.

Technical Specifications:

1. Use an abstract base class Employee with private attributes name and **employeeID**.
2. Declare a pure virtual method **displayInfo()** and a virtual method **isEqual(const Employee& other).**
3. Create **Manager**, **Developer**, and **Intern classes** that inherit publicly from Employee.
4. Manager adds private attribute department; Developer adds **programmingLanguage**; Intern adds **durationMonths**.
5. Override **displayInfo()** in each derived class.
6. Implement a function **identifyRole(Employee* emp)** that uses **dynamic_cast** to print the employee's role.
7. In **main(),** create at least one object of each derived class, call **identifyRole()** on each, and compare two employees using **isEqual().**
8. Use proper access specifiers (private, public).
9. Apply dynamic memory allocation (DMA) where appropriate.

Note:

Adding the following two methods to any derived class (Manager, Developer, or Intern) will lead to an extra **5% bonus** weightage:

- **promote()** — simulates promotion activities
- **evaluate()** — simulates performance evaluation

[15]

Section C – 80 Minutes

Complex Programs

Examiner Use

3 You are tasked with writing a C++ program for a **medical appointment booking system designed for a local hospital**. The system allows patients to book appointments, manage their information, and view scheduled appointments. The system is **designed using object-oriented programming principles** including inheritance, polymorphism, encapsulation, and dynamic memory allocation.

The system consists of the following classes:

- **HospitalLocation** – stores city and branch name.
- **MedicalRecord** – stores patient notes or diagnosis summaries.
- **Appointment** – a base class that stores patient name, CNIC, and contact number (as private) along with appointment verification status. It also provides methods to set and display appointment information.
- **DoctorAppointment** – a derived class that adds the doctor's name and purpose of visit as public attributes. It overrides the base class methods to handle appointment-specific details.

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

Technical Specifications:

1. Create a **base class Appointment** with private attributes: patientName, cnic, contactNumber, and a public attribute isVerified.
2. Create a derived class **DoctorAppointment** that inherits from Appointment and adds public attributes: doctorName and visitPurpose.
3. **Override** base class methods in **DoctorAppointment** to handle additional appointment details.
4. Use **dynamic memory allocation** to create an array of Appointment pointers, sized at runtime.
5. Implement a menu-driven interface with the following options:
    - Add a new doctor appointment (input all relevant details).
    - Display all appointments (including a unique Appointment ID, current date/time using <ctime>, and verification status).
    - Exit the program.
6. Use **rand()** to auto-generate a unique Appointment ID.
7. Use the **ctime library** to store and display the current date and time of appointment creation.
8. Use **virtual functions** in the base class to enable polymorphism.
9. Ensure proper memory deallocation using delete to prevent memory leaks.
10. Use correct access modifiers (private, public) to protect patient information.

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 patient 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 functionality of appointments.
5. If the system were to be expanded to handle Specialist Appointments in addition to Doctor Appointments, 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