SECTION A: ALGORITHM AND PROGRAMMING

1. Data Structure and Algorithms

```
BEGIN
   VAR B: BOOLEAN
   VAR A: INTEGER
   VAR C: INTEGER

   WRITE "Enter value for A:"
   READ A

   IF B THEN
      WRITE "Processing C"
      C
      WRITE "Restarting A"
      A
   ENDIF

   WRITE "Final value of A:"
   WRITE A
END
```

II)

```
BEGIN
   VAR E, F, G: BOOLEAN
   VAR D: INTEGER

   WRITE "Enter value for D:"
   READ D

   IF D THEN
      WRITE "Enter value for E:"
      READ E

      IF E THEN
         WRITE "Enter value for F:"
         READ F

         IF F THEN
```

```
            WRITE "H"
         ELSE
            WRITE "I"
         ENDIF
      ELSE
         WRITE "I"
      ENDIF
   ENDIF

   WRITE "B"
END
```

2. Big-Oh Notation (5 marks)

Rule of sums: $O(f+g)=O(f)+O(g)$
False. Correct: $O(f+g)=\max(O(f),O(g))$
Rule of products: $O(f \cdot g)=O(f) \cdot O(g)$

True.
Transitivity: If $g=O(f)$ and $h=O(f)$ then $g=O(h)$
False. Correct: If $g=O(f)$ and $f=O(h)$ then $g=O(h)$

II) STRUCTURE PROGRAMING

1. Rules for Declaring Variables in C (3 marks)

Variable names can contain letters, digits, and underscores (_).
They must begin with a letter or underscore (not a digit).
They cannot be C keywords (e.g., int, if, return).
They are case-sensitive (age ≠ Age).
No special characters (e.g., @, $) are allowed.

2. Four Ways to Declare a Function in C (5 marks)
No arguments, no return value

```
void greet() {
printf("Hello!"); }
```

With arguments, no return value

```
void sum(int a, int b) {
printf("%d", a + b); }
```

No arguments, with return value
int getNum() {
return 42; }
With arguments and return value
int add(int a, int b) {
return a + b; }

3. Size of int arr[10] in 32-bit and 64-bit Systems (2 marks)

32-bit system:
int = 4 bytes → arr[10] = 40 bytes (10 × 4).

64-bit system:
int = 4 bytes → arr[10] = 40 bytes.

4. Size of struct employee (2 marks)

Given:

```
struct employee {
   int id;       // 4B
   char name[20]; // 20B
   float salary;  // 4B
};
```

Total size: 4 + 20 + 4 = 28 bytes.

5. Output of the Program (2 marks)

Code:

```
#include <stdio.h>
int main() {
   int a = 1, b = 1, c;
   c = a++ + b;  // c = 1 (a++) + 1 (b) = 2, then a increments to 2
   printf("%d, %d", a, b);  // Output: 2, 1
}
```

Output:
2, 1
Explanation:
a++ is post-increment → uses original value (1) in the expression, then increments a to 2.
b remains 1.

6. What the Code Does (1 mark)
Code:

```
#include <stdio.h>
int main() {
   FILE *fp;
   fp = fopen("data.txt", "w");  // Opens "data.txt" in write mode
}
```

Action:
Creates/overwrites a file named data.txt for writing (no data is written in this snippet).
If the file exists, its contents are erased. If not, it is created.

III. OBJECT ORIENTED PROGRAMMING

1. Definitions (2 marks)
Inheritance is a fundamental OOP concept where a new class (called a derived or child class) inherits attributes and methods from an existing class (called a base or parent class). This promotes code reusability and a hierarchical relationship between classes. Modularity, on the other hand, refers to the practice of dividing a program into smaller, independent, and interchangeable modules (such as classes or functions). This enhances maintainability, readability, and scalability by isolating functionality into distinct units.

2. Class vs. Object (2 marks)
A class is a blueprint or template that defines the structure and behavior of objects. It specifies what attributes (data members) and methods (member functions) the objects will have. For example, a Car class might define attributes like color and speed, along with methods like accelerate() and brake(). An object, however, is an actual instance of a class. It is a concrete entity created from the class blueprint, with its own unique data. For instance, myCar could be an object of the Car class, with specific values like color = "red" and speed = 60.

3. Abstract Class vs. Concrete Class (2 marks)
An abstract class is designed to be a base class and cannot be instantiated on its own. It typically contains at least one pure virtual function (e.g., virtual void display() = 0;), which must be implemented by any derived class. Abstract classes are used to define interfaces and enforce structure in derived classes. A concrete class, in contrast, is a fully implemented class that can be instantiated.

4. Types of Inheritance
Inheritance in OOP can take several forms:

Single Inheritance: A derived class inherits from only one base class (e.g., Dog inherits from Animal).

Multiple Inheritance: A class inherits from more than one base class (e.g., HybridCar inherits from ElectricCar and GasCar).

Multilevel Inheritance: A chain of inheritance where a class is derived from another derived class (e.g., Animal → Mammal → Dog).

Hierarchical Inheritance: Multiple classes inherit from a single base class (e.g., Cat and Dog both inherit from Animal).

Hybrid Inheritance: A combination of multiple and multilevel inheritance (e.g., Animal → Mammal and Animal → Bird, then Bat inherits from both Mammal and Bird).

5. Dot (.) Operator in C++ (1 mark)
The dot operator (.) in C++ is used to access members (attributes or methods) of an object or structure. For example, if you have an object student1 of a Student class with an attribute name, you can access it as student1.name. Similarly, to call a method like displayInfo(), you would write student1.displayInfo(). The dot operator is essential for interacting with an object's properties and behaviors directly.

6a) OOP Concepts Implemented:
Encapsulation - Data (profession, age) and methods bundled in classes

Inheritance - MathsTeacher and Footballer inherit from Person

Abstraction - display() method shows only essential features

Polymorphism - (Not strongly shown, but potential through virtual functions)

b) Program Output:
My profession is: Teacher
My age is: 23
I can walk.
I can talk.
I can teach Maths.

My profession is: Footballer
My age is: 19
I can walk.
I can talk.
I can play Football.

SECTION B: DATABASE DEVELOPMENT AND ADMINISTRATION (20 MARKS)

EXERCISE IV (20 marks)

1. Field Data Types in CUSTOMER Table (8 marks)

i) Customer ID (Auto Numeric Field)

An auto-incremented numeric field that uniquely identifies each customer. The database automatically assigns a value, ensuring no duplicates. Used as a primary key for relationships.

ii) Customer Name (Text Field)

Stores alphabetic names of customers. Text fields allow variable-length strings for flexibility in inputting full names.

iii) Fee Paid (Decimal Field)

Stores monetary values with precision (e.g., 250.50). Decimal fields avoid rounding errors, critical for financial calculations.

iv) Pay Date (Date Field)

Records transaction dates (e.g., 2023-05-30). Enables date-based queries (e.g., monthly reports) and ensures valid date formats.

2. Database Normalization (12 marks)

a) First Normal Form (1NF) Relation (2 marks)

VEHICLE_OPERATOR Table (1NF eliminates repeating groups):

VehicleID	Description	Operator	Route	TariffPerMile
V1	Luxury	Polax	Grand Trail	100
V1	Luxury	Ubet	East Route	150
V2	Comfort	Polax	Grand Trail	45
V2	Comfort	Ubet	East Route	60
V2	Comfort	Minim	South Trunk	35

Key Change: Each row now has a unique combination of VehicleID and Operator.

b) Functional Dependencies & Candidate Key (3 marks)

- Functional Dependencies:
  - VehicleID → Description
  - Operator → Route
  - (VehicleID, Operator) → TariffPerMile
- Candidate Key: (VehicleID, Operator) (uniquely identifies each row).

c) Anomalies (3 marks)

- Insert Anomaly: Cannot add a new vehicle without assigning an operator (due to composite key).
- Delete Anomaly: Deleting an operator (e.g., Minim) removes all their vehicle records.
- Modification Anomaly: Updating a route (e.g., "Grand Trail") requires changes in multiple rows.

d) Relational Schema & Dependencies (3 marks

e) Normal Form (1 mark)

The relation is in 1NF (atomic values, no repeating groups)

SECTION C: WEB DESIGN (15 MARKS):

1. What is JavaScript used for? (1 mark)

JavaScript is primarily used to make web pages interactive by enabling dynamic content updates, form validation, animations, and asynchronous communication with servers (AJAX). It runs client-side in browsers but can also be used server-side with Node.js.

2. Is it possible to implement a web project without using JavaScript? (1 mark)

Yes, a basic web project can be implemented using just HTML and CSS for static content and styling. However, dynamic functionality like form validation or interactive elements would require JavaScript or server-side alternatives.

3. Process of deploying a web application (2 marks)

The deployment process involves: (1) purchasing a domain name and web hosting, (2) uploading files via FTP or Git, (3) setting up databases if needed, and (4) configuring DNS to point the domain to the hosting server.

4. Define PHP and development tools (1,2 marks)

PHP is a server-side scripting language used for dynamic web development. Essential tools include:

- A web server (Apache/Nginx)
- PHP interpreter
- Database (MySQL/MariaDB)
- Code editor (VS Code/PHPStorm)
- Version control (Git)

5. Login Form (HTML + CSS) (4 marks)

```
<html>
<head>
  <style>
    body { font-family: Arial; margin: 50px; }
    .auth-form { width: 300px; margin: auto; }
    input { width: 100%; padding: 8px; margin: 5px 0; }
    button { background:  color: white; padding: 10px; border: none; }
  </style>
</head>
<body>
  <div class="auth-form">
    <h2>AUTHENTICATION</h2>
    <form action="process.php" method="post">
      <label>Login:</label>
      <input type="text" name="login" required>
      <label>Password:</label>
      <input type="password" name="password" required>
      <button type="submit">Submit</button>
      <button type="reset">Cancel</button>
    </form>
  </div>
</body>
</html>
```

6. PHP Processing File (4 marks

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $login = htmlspecialchars($_POST['login']);
  $password = htmlspecialchars($_POST['password']);
  echo "Login: " . $login . "<br>";
  echo "Password: " . $password;
}
?>
```