CODE ASSIGNMENT OOP

UNIT 3

54. Class `Person` with Private Attributes and Getters/Setters

```
cpp
CopyEdit
#include <iostream>
using namespace std;

class Person {
private:
    string name;
    int age;

public:
    void setName(string n) { name = n; }
    void setAge(int a) { age = a; }
    string getName() { return name; }
    int getAge() { return age; }
};
```

---

55. Class `Student` Inheriting from `Person` with `studentID`

```
cpp
CopyEdit
class Student : public Person {
private:
    int studentID;

public:
    void setStudentID(int id) { studentID = id; }
    int getStudentID() { return studentID; }
};
```

---

56. Class `Car` with Display Method

```
cpp
CopyEdit
class Car {
private:
    string make;
    string model;
    int year;

public:
    Car(string m, string mo, int y) : make(m), model(mo), year(y) {}

    void display() {
```

```
        cout << "Make: " << make << ", Model: " << model << ", Year: " <<
year << endl;
    }
};
```

---

57. Array of `Car` Objects

```
cpp
CopyEdit
int main() {
    Car cars[] = {
        Car("Toyota", "Corolla", 2020),
        Car("Honda", "Civic", 2021),
        Car("Ford", "Mustang", 2019)
    };

    for (int i = 0; i < 3; ++i) {
        cars[i].display();
    }

    return 0;
}
```

---

58. Class `BankAccount` with Deposit/Withdraw

```
cpp
CopyEdit
class BankAccount {
private:
    string accountNumber;
    double balance;

public:
    BankAccount(string accNum, double initialBalance)
        : accountNumber(accNum), balance(initialBalance) {}

    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (amount <= balance)
            balance -= amount;
        else
            cout << "Insufficient funds!" << endl;
    }

    void display() {
        cout << "Account: " << accountNumber << ", Balance: $" << balance <<
endl;
    }
};
```

59. Class `Rectangle` with Area and Perimeter

```
cpp
CopyEdit
class Rectangle {
private:
    double length;
    double width;

public:
    Rectangle(double l, double w) : length(l), width(w) {}

    double area() { return length * width; }
    double perimeter() { return 2 * (length + width); }
};
```

---

60. Class `Employee` with Display Method

```
cpp
CopyEdit
class Employee {
private:
    string name;
    string position;
    double salary;

public:
    Employee(string n, string p, double s) : name(n), position(p), salary(s)
{}

    void display() {
        cout << "Name: " << name << ", Position: " << position << ", Salary:
$" << salary << endl;
    }
};
```

---

61. Class `Counter` with Static Object Counter

```
cpp
CopyEdit
class Counter {
private:
    static int count;

public:
    Counter() { ++count; }
    static int getCount() { return count; }
};

int Counter::count = 0;
```

---

62. Class `Math` with Static Arithmetic Methods

```
cpp
CopyEdit
class Math {
public:
    static int add(int a, int b) { return a + b; }
    static int subtract(int a, int b) { return a - b; }
    static int multiply(int a, int b) { return a * b; }
    static double divide(double a, double b) {
        return (b != 0) ? a / b : 0;
    }
};
```

---

63. Class `Student` with Static Member for Total Count

```
cpp
CopyEdit
class Student {
private:
    static int totalStudents;
    string name;

public:
    Student(string n) : name(n) { ++totalStudents; }

    static int getTotalStudents() { return totalStudents; }
};

int Student::totalStudents = 0;
```

64. Class `Book` with a Parameterized Constructor

```
cpp
CopyEdit
class Book {
private:
    string title;
    string author;
    double price;

public:
    Book(string t, string a, double p) : title(t), author(a), price(p) {}

    void display() {
        cout << "Title: " << title << ", Author: " << author << ", Price: $"
<< price << endl;
    }
};
```

---

65. Class `Point` with Default, Parameterized, and Copy Constructors

```
cpp
CopyEdit
class Point {
private:
    int x, y;

public:
    Point() : x(0), y(0) {}  // Default constructor
    Point(int a, int b) : x(a), y(b) {}  // Parameterized constructor
    Point(const Point &p) : x(p.x), y(p.y) {}  // Copy constructor

    void display() {
        cout << "(" << x << ", " << y << ")" << endl;
    }
};
```

---

66. Class `Matrix` with a Parameterized Constructor

```
cpp
CopyEdit
class Matrix {
private:
    int rows, cols;
    int** data;

public:
    Matrix(int r, int c) : rows(r), cols(c) {
        data = new int*[rows];
        for (int i = 0; i < rows; ++i)
            data[i] = new int[cols]{0};  // Initialize with zeros
    }

    void setElement(int r, int c, int value) {
        if (r < rows && c < cols)
            data[r][c] = value;
    }

    void display() {
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j)
                cout << data[i][j] << " ";
            cout << endl;
        }
    }

    ~Matrix() {
        for (int i = 0; i < rows; ++i)
            delete[] data[i];
        delete[] data;
    }
};
```

---

67. Class `FileHandler` with a Destructor to Close File

```
cpp
CopyEdit
#include <fstream>

class FileHandler {
private:
    fstream file;

public:
    FileHandler(string filename) {
        file.open(filename, ios::out);
        if (file.is_open())
            file << "File opened successfully.\n";
    }

    ~FileHandler() {
        if (file.is_open()) {
            file << "File closing.\n";
            file.close();
        }
    }
};
```

---

68. Class `DynamicArray` with Destructor to Deallocate Memory

```
cpp
CopyEdit
class DynamicArray {
private:
    int* arr;
    int size;

public:
    DynamicArray(int s) : size(s) {
        arr = new int[size];
    }

    void set(int index, int value) {
        if (index >= 0 && index < size)
            arr[index] = value;
    }

    void display() {
        for (int i = 0; i < size; ++i)
            cout << arr[i] << " ";
        cout << endl;
    }

    ~DynamicArray() {
        delete[] arr;
    }
};
```

69. Class `Logger` with Destructor Logging Message

```
cpp
CopyEdit
#include <iostream>
using namespace std;

class Logger {
private:
    string logMessage;

public:
    Logger(string msg) : logMessage(msg) {
        cout << "Logger created: " << logMessage << endl;
    }

    ~Logger() {
        cout << "Logger destroyed: " << logMessage << endl;
    }
};
```

---

70. Class `Complex` Overloading `+` Operator

```
cpp
CopyEdit
class Complex {
private:
    double real, imag;

public:
    Complex(double r = 0, double i = 0) : real(r), imag(i) {}

    Complex operator+(const Complex& other) {
        return Complex(real + other.real, imag + other.imag);
    }

    void display() {
        cout << real << " + " << imag << "i" << endl;
    }
};
```

---

71. Overloading `>>` and `<<` for Class `Fraction`

```
cpp
CopyEdit
#include <iostream>
using namespace std;

class Fraction {
private:
    int numerator, denominator;
```

```
public:
    Fraction() : numerator(0), denominator(1) {}

    friend ostream& operator<<(ostream& out, const Fraction& f) {
        out << f.numerator << "/" << f.denominator;
        return out;
    }

    friend istream& operator>>(istream& in, Fraction& f) {
        cout << "Enter numerator and denominator: ";
        in >> f.numerator >> f.denominator;
        return in;
    }
};
```

---

72. Overload `==` Operator for Class `Date`

```
cpp
CopyEdit
class Date {
private:
    int day, month, year;

public:
    Date(int d, int m, int y) : day(d), month(m), year(y) {}

    bool operator==(const Date& other) {
        return (day == other.day && month == other.month && year ==
other.year);
    }
};
```

---

73. Class `Vector` with Overloaded `[]` Operator

```
cpp
CopyEdit
class Vector {
private:
    int* elements;
    int size;

public:
    Vector(int s) : size(s) {
        elements = new int[size];
    }

    int& operator[](int index) {
        return elements[index];
    }

    ~Vector() {
        delete[] elements;
    }
```

```
};
```

74. Class `Box` with a Friend Function to Calculate Volume

```
cpp
CopyEdit
class Box {
private:
    double length, width, height;

public:
    Box(double l, double w, double h) : length(l), width(w), height(h) {}

    friend double calculateVolume(const Box& b1, const Box& b2);
};

double calculateVolume(const Box& b1, const Box& b2) {
    return (b1.length * b1.width * b1.height) + (b2.length * b2.width *
b2.height);
}
```

---

75. Class `Circle` with a Friend Function to Calculate Area

```
cpp
CopyEdit
#define PI 3.14159

class Circle {
private:
    double radius;

public:
    Circle(double r) : radius(r) {}

    friend double calculateArea(const Circle& c);
};

double calculateArea(const Circle& c) {
    return PI * c.radius * c.radius;
}
```

---

76. Class `Distance` with a Friend Function to Add Two Distances

```
cpp
CopyEdit
class Distance {
private:
    int feet, inches;

public:
    Distance(int f, int i) : feet(f), inches(i) {}
```

```
    friend Distance addDistance(const Distance& d1, const Distance& d2);
    void display() {
        cout << feet << " feet " << inches << " inches" << endl;
    }
};

Distance addDistance(const Distance& d1, const Distance& d2) {
    int totalInches = d1.inches + d2.inches;
    int totalFeet = d1.feet + d2.feet + totalInches / 12;
    totalInches %= 12;
    return Distance(totalFeet, totalInches);
}
```

---

77. Base Class `Shape` with Derived Classes `Circle`, `Rectangle`, and `Triangle`

```
cpp
CopyEdit
class Shape {
public:
    virtual void display() = 0; // Pure virtual function
};

class CircleShape : public Shape {
public:
    void display() override {
        cout << "This is a circle." << endl;
    }
};

class RectangleShape : public Shape {
public:
    void display() override {
        cout << "This is a rectangle." << endl;
    }
};

class TriangleShape : public Shape {
public:
    void display() override {
        cout << "This is a triangle." << endl;
    }
};
```

---

78. Base Class `Animal` with Derived Classes `Dog`, `Cat`, and `Bird`

```
cpp
CopyEdit
class Animal {
public:
    virtual void speak() {
        cout << "Animal sound" << endl;
    }
};
```

```
class Dog : public Animal {
public:
    void speak() override {
        cout << "Woof!" << endl;
    }
};

class Cat : public Animal {
public:
    void speak() override {
        cout << "Meow!" << endl;
    }
};

class Bird : public Animal {
public:
    void speak() override {
        cout << "Tweet!" << endl;
    }
};
```

79. Class `Vehicle` with Derived Classes `Car` and `Bike`

```
cpp
CopyEdit
class Vehicle {
public:
    virtual void showType() {
        cout << "This is a vehicle." << endl;
    }
};

class Car : public Vehicle {
public:
    void showType() override {
        cout << "This is a car." << endl;
    }
};

class Bike : public Vehicle {
public:
    void showType() override {
        cout << "This is a bike." << endl;
    }
};
```

---

80. Single Inheritance: Base Class `Person`, Derived Class `Employee`

```
cpp
CopyEdit
class Person {
protected:
    string name;
```

```
public:
    void setName(string n) { name = n; }
};

class Employee : public Person {
private:
    string position;

public:
    void setPosition(string p) { position = p; }

    void display() {
        cout << "Name: " << name << ", Position: " << position << endl;
    }
};
```

---

81. Multiple Inheritance: Class `Parent`, Derived Classes `Child1` and `Child2`

```
cpp
CopyEdit
class Parent {
public:
    void show() {
        cout << "Parent class" << endl;
    }
};

class Child1 : public Parent {
public:
    void child1Func() {
        cout << "Child1 class" << endl;
    }
};

class Child2 : public Parent {
public:
    void child2Func() {
        cout << "Child2 class" << endl;
    }
};
```

---

82. Hierarchical Inheritance: `Base` → `Derived1`, `Derived2`, `Derived3`

```
cpp
CopyEdit
class Base {
public:
    void display() {
        cout << "Base class" << endl;
    }
};
```

```
class Derived1 : public Base {
public:
    void d1() {
        cout << "Derived1 class" << endl;
    }
};

class Derived2 : public Base {
public:
    void d2() {
        cout << "Derived2 class" << endl;
    }
};

class Derived3 : public Base {
public:
    void d3() {
        cout << "Derived3 class" << endl;
    }
};
```

---

83. Multilevel Inheritance: `Base` → `Intermediate` → `Derived`

```
cpp
CopyEdit
class Base {
public:
    void baseFunc() {
        cout << "Base class" << endl;
    }
};

class Intermediate : public Base {
public:
    void interFunc() {
        cout << "Intermediate class" << endl;
    }
};

class Derived : public Intermediate {
public:
    void derivedFunc() {
        cout << "Derived class" << endl;
    }
};
```

84. Hybrid Inheritance: Base → Derived1, Derived2 → Derived3

```
cpp
CopyEdit
class Base {
public:
    void show() {
        cout << "Base class" << endl;
```

```
    }
};

class Derived1 : virtual public Base {
public:
    void showD1() {
        cout << "Derived1 class" << endl;
    }
};

class Derived2 : virtual public Base {
public:
    void showD2() {
        cout << "Derived2 class" << endl;
    }
};

class Derived3 : public Derived1, public Derived2 {
public:
    void showD3() {
        cout << "Derived3 class (Hybrid Inheritance)" << endl;
    }
};
```

---

85. Class `Library` with Private, Protected, and Public Members

```
cpp
CopyEdit
class Library {
private:
    string librarian;

protected:
    int totalBooks;

public:
    void setLibrarian(string name) { librarian = name; }
    void setBooks(int books) { totalBooks = books; }
    void display() {
        cout << "Librarian: " << librarian << ", Total books: " << totalBooks
<< endl;
    }
};
```

---

86. Class `Account` with Encapsulation

```
cpp
CopyEdit
class Account {
private:
    string owner;
    double balance;
```

```
public:
    void setOwner(string o) { owner = o; }
    void setBalance(double b) { balance = b; }
    string getOwner() { return owner; }
    double getBalance() { return balance; }
};
```

---

87. Function Overriding: `Shape` and `Circle`

```
cpp
CopyEdit
class Shape {
public:
    virtual void draw() {
        cout << "Drawing a shape." << endl;
    }
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing a circle." << endl;
    }
};
```

---

88. Base Class `Employee` and Derived Class `Manager` with Overridden Methods

```
cpp
CopyEdit
class Employee {
public:
    virtual void work() {
        cout << "Employee working..." << endl;
    }
};

class Manager : public Employee {
public:
    void work() override {
        cout << "Manager supervising work..." << endl;
    }
};
```

89. Virtual Base Class `Entity` with Derived Classes `Person` and `Organization`

```
cpp
CopyEdit
class Entity {
public:
    virtual void showEntity() = 0; // Pure virtual function
};
```

```
class Person : virtual public Entity {
public:
    void showEntity() override {
        cout << "This is a person." << endl;
    }
};

class Organization : virtual public Entity {
public:
    void showEntity() override {
        cout << "This is an organization." << endl;
    }
};
```

---

90. Class `Animal` with Virtual Base Class to Avoid Diamond Problem

```
cpp
CopyEdit
class Animal {
public:
    virtual void sound() {
        cout << "Animal sound" << endl;
    }
};

class Mammal : virtual public Animal {
public:
    void sound() override {
        cout << "Mammal sound" << endl;
    }
};

class Bird : virtual public Animal {
public:
    void sound() override {
        cout << "Bird chirp" << endl;
    }
};

class Bat : public Mammal, public Bird {
public:
    void sound() override {
        cout << "Bat sound" << endl;
    }
};
```

---

91. Class `Polynomial` with Functions to Add and Multiply Polynomials

```
cpp
CopyEdit
class Polynomial {
private:
    int a, b, c;  // Representing polynomial as ax^2 + bx + c
```

```
public:
    Polynomial(int x, int y, int z) : a(x), b(y), c(z) {}

    Polynomial add(const Polynomial& p) {
        return Polynomial(a + p.a, b + p.b, c + p.c);
    }

    Polynomial multiply(const Polynomial& p) {
        int newA = a * p.a;
        int newB = a * p.b + b * p.a;
        int newC = b * p.b + c * p.a + c * p.b;
        return Polynomial(newA, newB, newC);
    }

    void display() {
        cout << a << "x^2 + " << b << "x + " << c << endl;
    }
};
```

---

92. Class `SparseMatrix` with Addition and Multiplication

```
cpp
CopyEdit
class SparseMatrix {
private:
    int rows, cols;
    vector<vector<int>> matrix;

public:
    SparseMatrix(int r, int c) : rows(r), cols(c) {
        matrix.resize(rows, vector<int>(cols, 0));
    }

    void setElement(int r, int c, int value) {
        matrix[r][c] = value;
    }

    SparseMatrix add(const SparseMatrix& other) {
        SparseMatrix result(rows, cols);
        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < cols; ++j)
                result.setElement(i, j, matrix[i][j] + other.matrix[i][j]);
        return result;
    }

    void display() {
        for (auto& row : matrix) {
            for (int val : row)
                cout << val << " ";
            cout << endl;
        }
    }
};
```

93. Class `Time` with Functions to Add, Subtract, and Compare Time

```
cpp
CopyEdit
class Time {
private:
    int hours, minutes, seconds;

public:
    Time(int h, int m, int s) : hours(h), minutes(m), seconds(s) {}

    Time add(const Time& t) {
        int totalSeconds = (hours * 3600 + minutes * 60 + seconds) +
                           (t.hours * 3600 + t.minutes * 60 + t.seconds);
        return Time(totalSeconds / 3600, (totalSeconds % 3600) / 60,
totalSeconds % 60);
    }

    Time subtract(const Time& t) {
        int totalSeconds = (hours * 3600 + minutes * 60 + seconds) -
                           (t.hours * 3600 + t.minutes * 60 + t.seconds);
        return Time(totalSeconds / 3600, (totalSeconds % 3600) / 60,
totalSeconds % 60);
    }

    bool compare(const Time& t) {
        return (hours == t.hours && minutes == t.minutes && seconds ==
t.seconds);
    }

    void display() {
        cout << hours << ":" << minutes << ":" << seconds << endl;
    }
};
```

94. Class `BigNumber` to Handle Arithmetic Operations on Large Numbers

```
cpp
CopyEdit
#include <iostream>
#include <string>
using namespace std;

class BigNumber {
private:
    string number;

public:
    BigNumber(string num) : number(num) {}

    // Adding two big numbers (simple string-based approach)
    BigNumber add(const BigNumber& other) {
        string result = "";
```

```
        int carry = 0;
        int i = number.length() - 1;
        int j = other.number.length() - 1;

        while (i >= 0 || j >= 0 || carry) {
            int sum = carry;
            if (i >= 0) sum += number[i] - '0';
            if (j >= 0) sum += other.number[j] - '0';
            result = (char)(sum % 10 + '0') + result;
            carry = sum / 10;
            i--; j--;
        }

        return BigNumber(result);
    }

    void display() {
        cout << number << endl;
    }
};
```

---

95. Class `FileCompressor` with Methods to Compress and Decompress Files

```
cpp
CopyEdit
#include <iostream>
#include <fstream>
using namespace std;

class FileCompressor {
public:
    void compress(const string& filename) {
        // In a real program, this would compress the file content.
        cout << "Compressing file: " << filename << endl;
    }

    void decompress(const string& filename) {
        // In a real program, this would decompress the file content.
        cout << "Decompressing file: " << filename << endl;
    }
};
```

---

96. Class `Network` to Simulate Network Packet Transmission

```
cpp
CopyEdit
#include <iostream>
using namespace std;

class Network {
public:
    void sendPacket(const string& packet) {
        cout << "Sending packet: " << packet << endl;
```

```
    }

    void receivePacket(const string& packet) {
        cout << "Received packet: " << packet << endl;
    }
};
```

---

97. Class `Cache` to Store and Retrieve Cached Data

```
cpp
CopyEdit
#include <iostream>
#include <unordered_map>
using namespace std;

class Cache {
private:
    unordered_map<string, string> cache;

public:
    void storeData(const string& key, const string& value) {
        cache[key] = value;
        cout << "Data stored: " << key << " -> " << value << endl;
    }

    void retrieveData(const string& key) {
        if (cache.find(key) != cache.end()) {
            cout << "Data retrieved: " << key << " -> " << cache[key] <<
endl;
        } else {
            cout << "Data not found for key: " << key << endl;
        }
    }
};
```

---

98. Class `Game` to Simulate a Simple Game with Player Actions and Scoring

```
cpp
CopyEdit
#include <iostream>
using namespace std;

class Game {
private:
    int score;

public:
    Game() : score(0) {}

    void playerAction(const string& action) {
        if (action == "score") {
            score += 10;
            cout << "Player scored! Current score: " << score << endl;
```

```
        } else if (action == "penalty") {
            score -= 5;
            cout << "Player got a penalty! Current score: " << score << endl;
        } else {
            cout << "Invalid action." << endl;
        }
    }

    void displayScore() {
        cout << "Current score: " << score << endl;
    }
};
```