UID _ 22BCS14121 BAVNEET KAUR

```
// 1) Sum of Natural Numbers up to N
#include<iostream>
using namespace std;
int sumNaturalNumbers(int n) {
    return n * (n + 1) / 2;
}

// 2) Check if a Number is Prime
bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) return false;
    return true;
}

// 3) Print Odd Numbers up to N
void printOddNumbers(int n) {
    for (int i = 1; i <= n; i += 2)
        cout << i << " ";
    cout << endl;
}

// 4) Sum of Odd Numbers up to N
int sumOddNumbers(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i += 2)
        sum += i;
    return sum;
}

// 5) Print Multiplication Table of a Number
void printMultiplicationTable(int num) {
    for (int i = 1; i <= 10; ++i)
        cout << num << " x " << i << " = " << num * i << endl;
}

// 6) Count Digits in a Number
int countDigits(int num) {
    int count = 0;
    while (num > 0) {
        num /= 10;
        ++count;
```

```
    }
    return count;
}

// 7) Find the Largest Digit in a Number
int largestDigit(int num) {
    int largest = 0;
    while (num > 0) {
        largest = max(largest, num % 10);
        num /= 10;
    }
    return largest;
}

// 8) Find the Sum of Digits of a Number
int sumOfDigits(int num) {
    int sum = 0;
    while (num > 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

// 9) Function Overloading for Calculating Area
#include <cmath>
double area(double radius) {
    return M_PI * radius * radius;
}
double area(double length, double breadth) {
    return length * breadth;
}
double area(double base, double height, bool isTriangle) {
    return 0.5 * base * height;
}

// 10) Function Overloading with Hierarchical Structure
class Hierarchical {
public:
    void print(int x) { cout << "Integer: " << x << endl; }
    void print(double y) { cout << "Double: " << y << endl; }
    void print(const string &z) { cout << "String: " << z << endl; }
};
```

```
// 11) Encapsulation with Employee Details
class Employee {
private:
    string name;
    int age;
    double salary;
public:
    void setDetails(string n, int a, double s) {
        name = n; age = a; salary = s;
    }
    void displayDetails() {
        cout << "Name: " << name << ", Age: " << age << ", Salary: " << salary << endl;
    }
};

// 12) Polymorphism with Shape Area Calculation
class Shape {
public:
    virtual double calculateArea() const = 0;
};
class Circle : public Shape {
    double radius;
public:
    Circle(double r) : radius(r) {}
    double calculateArea() const override { return M_PI * radius * radius; }
};
class Rectangle : public Shape {
    double length, breadth;
public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    double calculateArea() const override { return length * breadth; }
};

// 13) Implementing Polymorphism for Shape Hierarchies
// See 12 (reused structure)

// 14) Matrix Multiplication Using Function Overloading
void multiplyMatrices(int a[2][2], int b[2][2], int res[2][2]) {
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++) {
            res[i][j] = 0;
            for (int k = 0; k < 2; k++)
                res[i][j] += a[i][k] * b[k][j];
        }
```

```
}

// 15) Polymorphism in Shape Classes
// See 12 (reused structure)

// 16) Implement Multiple Inheritance to Simulate a Library System
class Book {
protected:
    string title;
public:
    void setBookTitle(string t) { title = t; }
};
class Member {
protected:
    string memberName;
public:
    void setMemberName(string name) { memberName = name; }
};
class Library : public Book, public Member {
public:
    void issueBook() {
        cout << memberName << " issued the book titled " << title << endl;
    }
};

// 17) Implement Polymorphism for Banking Transactions
class BankTransaction {
public:
    virtual void performTransaction() const = 0;
};
class Deposit : public BankTransaction {
    void performTransaction() const override {
        cout << "Depositing money..." << endl;
    }
};
class Withdraw : public BankTransaction {
    void performTransaction() const override {
        cout << "Withdrawing money..." << endl;
    }
};

// 18) Hierarchical Inheritance for Employee Management System
class Person {
protected:
```

```
    string name;
public:
    void setName(string n) { name = n; }
};
class Manager : public Person {
public:
    void display() { cout << name << " is a Manager." << endl; }
};
class Worker : public Person {
public:
    void display() { cout << name << " is a Worker." << endl; }
};

// 19) Multi-Level Inheritance for Vehicle Simulation
class Vehicle {
protected:
    string brand;
public:
    void setBrand(string b) { brand = b; }
};
class Car : public Vehicle {
protected:
    string model;
public:
    void setModel(string m) { model = m; }
};
class ElectricCar : public Car {
public:
    void display() { cout << brand << " " << model << " is an Electric Car." << endl; }
};

// 20) Function Overloading for Complex Number Operations
#include<complex>
complex<double> add(complex<double> a, complex<double> b) {
    return a + b;
}
complex<double> multiply(complex<double> a, complex<double> b) {
    return a * b;
}

int main() {
    // Test all functions/classes here as needed
    return 0;
}
```