ASSIGNMENT NO.1

```
#include <iostream>
#include <cmath>

using namespace std;

class AreaCalculator {
public:
    // Function to calculate the area of a triangle
    double area(double base, double height) {
        return 0.5 * base * height;
    }

    // Function to calculate the area of a circle
    double area(double radius) {
        return M_PI * radius * radius;
    }
};

int main() {
    AreaCalculator calculator;

    // Calculate and display the area of a triangle
    double triangleBase, triangleHeight;
    cout << "Enter the base of the triangle: ";
    cin >> triangleBase;
    cout << "Enter the height of the triangle: ";
    cin >> triangleHeight;
    cout << "Area of the triangle: " << calculator.area(triangleBase, triangleHeight) << endl;

    // Calculate and display the area of a circle
    double circleRadius;
    cout << "Enter the radius of the circle: ";
    cin >> circleRadius;
    cout << "Area of the circle: " << calculator.area(circleRadius) << endl;

    return 0;
}
```

ASSIGNMENT NNO.2

```
#include <iostream>
#include <iomanip>

using namespace std;

class BankAccount {
private:
    string depositorName;
    long accountNumber;
    char accountType;
    double balance;

public:
    // Function to assign initial values
    void initializeAccount(string name, long accNumber, char accType, double initialBalance) {
        depositorName = name;
        accountNumber = accNumber;
        accountType = accType;
        balance = initialBalance;
    }
    // Function to deposit an amount
    void deposit(double amount) {
        balance += amount;
        cout << "Deposit successful. New balance: $" << fixed << setprecision(2) << balance <<
endl;
    }

    // Function to withdraw an amount after checking the balance
    void withdraw(double amount) {
        if (amount > balance) {
            cout << "Insufficient funds. Withdrawal failed.\n";
        } else {
            balance -= amount;
            cout << "Withdrawal successful. New balance: $" << fixed << setprecision(2) <<
balance << endl;
        }
    }

    // Function to display name and balance
    void displayInfo() {
        cout << "Depositor Name: " << depositorName << endl;
        cout << "Account Number: " << accountNumber << endl;
        cout << "Account Type: " << accountType << endl;
        cout << "Current Balance: $" << fixed << setprecision(2) << balance << endl;
    }
};

int main() {
    // Creating an object of BankAccount
    BankAccount myAccount;

    // Initializing account details
    myAccount.initializeAccount("John Doe", 123456789, 'S', 1000.0);

    // Displaying initial information
    cout << "Initial Account Information:\n";
    myAccount.displayInfo();

    // Depositing and withdrawing money
    myAccount.deposit(500.0);
    myAccount.withdraw(200.0);
    myAccount.withdraw(1500.0); // This should fail due to insufficient funds

    // Displaying final information
    cout << "\nFinal Account Information:\n";
    myAccount.displayInfo();

    return 0;
}
```

ASSIGNMENT NO. 3

```
#include <iostream>

class DB; // Forward declaration

class DM {
private:
    int meters;
    int centimeters;

public:
    DM(int m = 0, int cm = 0) : meters(m), centimeters(cm) {}
    // Friend function declaration
    friend DM addDistance(DM, DB);

    void display() {
        std::cout << "Distance in meters and centimeters: " << meters << "m " << centimeters <<
"cm\n";
    }
};

class DB {
private:
    int feet;
    int inches;

public:
    DB(int ft = 0, int in = 0) : feet(ft), inches(in) {}

    // Friend function declaration
    friend DM addDistance(DM, DB);

    void display() {
        std::cout << "Distance in feet and inches: " << feet << "ft " << inches << "in\n";
    }
};

// Friend function definition
DM addDistance(DM dm, DB db) {
    int totalMeters = dm.meters + static_cast<int>(0.3048 * db.feet); // 1 foot = 0.3048 meters
    int totalCentimeters = dm.centimeters + static_cast<int>(2.54 * db.inches); // 1 inch = 2.54
centimeters

    // Handling overflow
    if (totalCentimeters >= 100) {
        totalMeters += totalCentimeters / 100;
        totalCentimeters %= 100;
    }

    return DM(totalMeters, totalCentimeters);
}

int main() {
    // Input values for DM object
    int m, cm;
    std::cout << "Enter distance in meters and centimeters:\n";
    std::cout << "Meters: ";
    std::cin >> m;
    std::cout << "Centimeters: ";
    std::cin >> cm;

    DM dmObj(m, cm);

    // Input values for DB object
    int ft, in;
    std::cout << "Enter distance in feet and inches:\n";
    std::cout << "Feet: ";
    std::cin >> ft;
    std::cout << "Inches: ";
    std::cin >> in;

    DB dbObj(ft, in);

    // Add DM object with DB object using friend function
    DM result = addDistance(dmObj, dbObj);
    // Display the result
    std::cout << "\nResult after addition:\n";
    result.display();

    return 0;
}
```

ASSIGNMENT NO. 4

```
#include <iostream>
#include <vector>

class MAT {
private:
    int rows;
    int cols;
    std::vector<std::vector<int>> matrix;

public:
    MAT(int m, int n) : rows(m), cols(n), matrix(m, std::vector<int>(n, 0)) {}

    void inputMatrix() {
        std::cout << "Enter matrix elements:\n";
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < cols; ++j) {
                std::cout << "Enter element at position (" << i + 1 << ", " << j + 1 << "): ";
                std::cin >> matrix[i][j];
            }
        }
    }
```

```
void displayMatrix() const {
    std::cout << "Matrix:\n";
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }
}

MAT add(const MAT& other) const {
    if (rows != other.rows || cols != other.cols) {
        std::cerr << "Matrix addition is not possible. Dimensions mismatch.\n";
        return MAT(0, 0); // Returning an empty matrix
    }

    MAT result(rows, cols);

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            result.matrix[i][j] = matrix[i][j] + other.matrix[i][j];
        }
    }

    return result;
}

MAT multiply(const MAT& other) const {
    if (cols != other.rows) {
        std::cerr << "Matrix multiplication is not possible. Inner dimensions do not match.\n";
        return MAT(0, 0); // Returning an empty matrix
        }

        MAT result(rows, other.cols);

        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < other.cols; ++j) {
                for (int k = 0; k < cols; ++k) {
                    result.matrix[i][j] += matrix[i][k] * other.matrix[k][j];
                }
            }
        }

        return result;
    }
};

int main() {
    int m1, n1, m2, n2;

    std::cout << "Enter dimensions of the first matrix (m1 n1): ";
    std::cin >> m1 >> n1;

    std::cout << "Enter dimensions of the second matrix (m2 n2): ";
    std::cin >> m2 >> n2;

    MAT mat1(m1, n1);
    mat1.inputMatrix();

    MAT mat2(m2, n2);
    mat2.inputMatrix();
    MAT sum = mat1.add(mat2);
    MAT product = mat1.multiply(mat2);

    std::cout << "\nMatrix 1:\n";
    mat1.displayMatrix();

    std::cout << "\nMatrix 2:\n";
    mat2.displayMatrix();

    std::cout << "\nSum of matrices:\n";
    sum.displayMatrix();

    std::cout << "\nProduct of matrices:\n";
    product.displayMatrix();

    return 0;
}
```

ASSIGNMENT NO.5

```
#include <iostream>
#include <cstring>

class Stud {
private:
    char* name;
    int age;
    float gpa;

public:
    // Default Constructor
    Stud() : name(nullptr), age(0), gpa(0.0) {
```

```
    std::cout << "Default Constructor called.\n";
}

// Multiple Constructor
Stud(const char* n, int a, float g) : age(a), gpa(g) {
    name = new char[strlen(n) + 1];
    strcpy(name, n);
    std::cout << "Multiple Constructor called.\n";
}

// Copy Constructor
Stud(const Stud& other) : age(other.age), gpa(other.gpa) {
    name = new char[strlen(other.name) + 1];
    strcpy(name, other.name);
    std::cout << "Copy Constructor called.\n";
}

// Overloaded Constructor
Stud(int a) : age(a), gpa(0.0), name(nullptr) {
    std::cout << "Overloaded Constructor called.\n";
}

// Destructor
~Stud() {
    delete[] name;
    std::cout << "Destructor called.\n";
}

// Display Student Information
void displayInfo() const {
    std::cout << "Name: " << (name ? name : "N/A") << std::endl;
```

```
        std::cout << "Age: " << age << std::endl;
        std::cout << "GPA: " << gpa << std::endl;
    }
};

int main() {
    // Default Constructor
    Stud stud1;

    // Multiple Constructor
    Stud stud2("Alice", 20, 3.8);

    // Copy Constructor
    Stud stud3 = stud2;

    // Overloaded Constructor
    Stud stud4(22);

    std::cout << "\nStudent Information:\n";
    stud1.displayInfo();
    std::cout << "\nStudent Information:\n";
    stud2.displayInfo();
    std::cout << "\nStudent Information (Copy):\n";
    stud3.displayInfo();
    std::cout << "\nStudent Information:\n";
    stud4.displayInfo();

    return 0;
}
```

ASSIGNMENT NO. 6

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

// Base class Person
class Person {
protected:
    string name;
    int age;

public:
    Person(const string& n, int a) : name(n), age(a) {}

    void display() const {
        cout << "Name: " << name << ", Age: " << age;
    }
};

// Derived class Account from Person
class Account : public Person {
protected:
    string accountNumber;

public:
    Account(const string& n, int a, const string& accNum)
        : Person(n, a), accountNumber(accNum) {}

    void displayAccount() const {
        display();
        cout << ", Account Number: " << accountNumber << endl;
    }
};

// Derived class Admin from Person
class Admin : public Person {
protected:
    string adminID;

public:
    Admin(const string& n, int a, const string& adminId)
        : Person(n, a), adminID(adminId) {}

    void displayAdmin() const {
        display();
        cout << ", Admin ID: " << adminID << endl;
    }
};

// Derived class Master from both Account and Admin
class Master : public Account, public Admin {
public:
    Master(const string& n, int a, const string& accNum, const string& adminId)
        : Account(n, a, accNum), Admin(n, a, adminId) {}

    void displayMaster() const {
        displayAccount();
        cout << ", ";
        displayAdmin();
    }
};
```

```
int main() {
    // Create a Master object
    Master master("John Doe", 30, "123456", "admin123");

    // Display information contained in the Master object
    cout << "Master Information:" << endl;
    master.displayMaster();

    // Update information
    master = Master("Jane Smith", 25, "654321", "admin456");

    // Display updated information
    cout << "\nUpdated Master Information:" << endl;
    master.displayMaster();

    return 0;
}
```

ASSINGNMENT NO. 7

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

// Base class Media
class Media {
protected:
    string title;
    float price;

public:
    Media(const string& t, float p) : title(t), price(p) {}

    // Virtual function for displaying media information
    virtual void display() const {
        cout << "Title: " << title << ", Price: $" << price;
    }
};

// Derived class for storing number of pages in the book
class Book : public Media {
private:
    int numPages;

public:
    Book(const string& t, float p, int pages) : Media(t, p), numPages(pages) {}

    // Override the display function for books
    void display() const override {
        Media::display();
        cout << ", Pages: " << numPages << endl;
    }
};

// Derived class for storing playing time of tape
class VideoTape : public Media {
private:
    float playingTime;

public:
    VideoTape(const string& t, float p, float time) : Media(t, p), playingTime(time) {}
```

```cpp
    // Override the display function for video tapes
    void display() const override {
        Media::display();
        cout << ", Playing Time: " << playingTime << " minutes" << endl;
    }
};

int main() {
    // Creating objects using polymorphism
    Media* items[3];
    items[0] = new Book("Book Title", 19.99, 300);
    items[1] = new VideoTape("Video Tape Title", 29.99, 120);
    items[2] = new Book("Another Book", 15.50, 200);

    // Displaying information using polymorphism
    cout << "Media Information:" << endl;
    for (int i = 0; i < 3; ++i) {
        items[i]->display();
    }

    // Cleaning up dynamically allocated memory
    for (int i = 0; i < 3; ++i) {
        delete items[i];
    }

    return 0;
}
```

ASSSIGNMENT NO. 7

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

class Sample {
private:
    int data;

public:
    // Constructor to initialize data
    Sample(int value) : data(value) {}

    // Function to set data using the this pointer
    void setData(int value) {
        this->data = value;
    }

    // Function to display data using the this pointer
    void displayData() const {
        cout << "Data: " << this->data << endl;
    }

    // Destructor to display a message when an object is deleted
    ~Sample() {
        cout << "Object with data " << this->data << " is deleted." << endl;
    }
};

int main() {
    // Creating objects dynamically using new
    Sample* obj1 = new Sample(42);
    Sample* obj2 = new Sample(99);

    // Displaying initial data
    cout << "Initial data:" << endl;
    obj1->displayData();
    obj2->displayData();

    // Using the this pointer to set data
    obj1->setData(100);
    obj2->setData(50);

    // Displaying updated data
    cout << "\nUpdated data:" << endl;
    obj1->displayData();
    obj2->displayData();

    // Deallocating memory using delete
    delete obj1;
    delete obj2;

    return 0;
}
```

ASSIGNMENT NO. 8

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

class Sample {
private:
    int data;
```

```
public:
    // Constructor to initialize data
    Sample(int value) : data(value) {}

    // Function to set data using the this pointer
    void setData(int value) {
        this->data = value;
    }

    // Function to display data using the this pointer
    void displayData() const {
        cout << "Data: " << this->data << endl;
    }

    // Destructor to display a message when an object is deleted
    ~Sample() {
        cout << "Object with data " << this->data << " is deleted." << endl;
    }
};

int main() {
    // Creating objects dynamically using new
    Sample* obj1 = new Sample(42);
    Sample* obj2 = new Sample(99);

    // Displaying initial data
    cout << "Initial data:" << endl;
    obj1->displayData();
    obj2->displayData();
    // Using the this pointer to set data
    obj1->setData(100);
    obj2->setData(50);

    // Displaying updated data
    cout << "\nUpdated data:" << endl;
    obj1->displayData();
    obj2->displayData();

    // Deallocating memory using delete
    delete obj1;
    delete obj2;

    return 0;
}
```

ASSSIGNMENT NO. 9

```
#include <iostream>

// Function template to find the minimum value in an array
template <typename T, size_t N>
T findMin(const T (&arr)[N]) {
    T minValue = arr[0];

    for (size_t i = 1; i < N; ++i) {
        if (arr[i] < minValue) {
            minValue = arr[i];
        }
    }
    return minValue;
}

int main() {
    // Example with an array of integers
    int intArray[] = {3, 7, 1, 9, 5};
    int minIntValue = findMin(intArray);
    std::cout << "Minimum value in the integer array: " << minIntValue << std::endl;

    // Example with an array of doubles
    double doubleArray[] = {2.5, 1.7, 3.8, 1.2, 4.5};
    double minDoubleValue = findMin(doubleArray);
    std::cout << "Minimum value in the double array: " << minDoubleValue << std::endl;

    return 0;
}
```

ASSIGNMENT NO. 10

```
#include <iostream>

int divide(int numerator, int denominator) {
    if (denominator == 0) {
        throw std::runtime_error("Error: Division by zero is not allowed.");
    }

    return numerator / denominator;
}

int main() {
    int numerator, denominator;
    // Get user input
    std::cout << "Enter numerator: ";
    std::cin >> numerator;

    std::cout << "Enter denominator: ";
    std::cin >> denominator;

    try {
        // Attempt to perform division and handle potential exception
        int result = divide(numerator, denominator);
        std::cout << "Result of division: " << result << std::endl;
    } catch (const std::exception& e) {
        // Handle the exception
        std::cerr << "Exception caught: " << e.what() << std::endl;
    }

    return 0;
}
```