ASSGNMENT 1 (area of triangle and circle)

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

class Area {
    public:
    float area(float base, float height){
        return 0.5 * base * height;
    }

    float area(float radius){
        return 3.14 * radius * radius;
    }
};

int main() {
    Area a;
    float base, height, radius;

    cout<<"Enter the base & height of the triangle : ";
    cin>>base>>height;
    cout<<"Area of triangle : "<<a.area(base, height)<<endl;

    cout<<"Enter the rradius of the circle : ";
    cin>>radius;
    cout<<"Area of circle : "<<a.area(radius)<<endl;

    return 0;
}
```

ASSIGNMENT 2 (bank account)

```cpp
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;

class bank
{
        int acc_number;
        char person_name[100], acc_type[100];
        float acc_balance;

    public :
        bank (int accno, char *name, char *acctyp, float balance)
        {
                acc_number = accno;
                strcpy (person_name, name);
                strcpy (acc_type, acctyp);
                acc_balance = balance;
        }
        void deposit();
        void withdraw();
        void display();
};

void bank :: deposit()
{
        int deposit_ammount;
        cout<<"\n Enter deposit amount : ";
        cin>>deposit_ammount;
        acc_balance += deposit_ammount;
```

```
}

void bank :: withdraw()
{
        int withdraw_ammount;
        cout<<"\n Enter withdraw amount : ";
        cin>>withdraw_ammount;
        acc_balance -= withdraw_ammount;
}

void bank :: display()
{
        cout<<"_____________________________________"<<endl;
        cout<<"\nACCOUNT NUMBER : "<<acc_number<<endl;
        cout<<"NAME : "<<person_name<<endl;
        cout<<"ACCOUNT TYPE : "<<acc_type<<endl;
        cout<<"BALANCE : "<<acc_balance<<endl;
}

int main() {
    int accno;
    char name[100], acctyp[100];
    float balance;
    cout<<"**ENTER DETAILS**"<<endl;
    cout<<"_____________________________________";
    cout<<"\n ACCOUNT NUMBER : ";
    cin>>accno;
    cout<<"\n NAME : ";
    cin>>name;
    cout<<"\n ACCOUNT TYPE : ";
    cin>>acctyp;
    cout<<"\n BALANCE : ";
    cin>>balance;

    bank b(accno, name, acctyp, balance);
    b.deposit();
    b.withdraw();
    b.display();
}
```

ASSIGNMENT 03 (classes DM and DB)

```
#include <iostream>
using namespace std;
class DB;
class  DM
{
private:
    int meters;
    float centimeters;

public:
    void  getdata()
    {
        cout << "Enter distance in meters:";
        cin >> meters;
        cout << "Enter distance in centimeters:";
        cin >> centimeters;
    }
    friend void add(DM, DB);
};
class DB
{
```

```
public:
    int feet;
    float inches;

public:
    void getdata()
    {
        cout << "Enter distance in feet:";
        cin >> feet;
        cout << "Enter distance in inches:";
        cin >> inches;
    }
    friend void add(DM, DB);
};
void add(DM dm, DB db)
{
    float total_meters = dm.meters + db.feet * 0.3048;
    float total_centimeters = dm.centimeters + db.inches * 2.54;
    if (total_centimeters >= 100)
    {
        int extra_meters = total_centimeters / 100;
        total_meters += extra_meters;
        total_centimeters -= extra_meters * 100;
    }
    cout << "Sum of distances is : " << total_meters << " meters and " << total_centimeters <<
" centimeters." << endl;
}
int main()
{
    DM dm;
    DB db;
    cout << "Enter the distance in meters and centimeters:" << endl;
    dm.getdata();
    cout << "Enter the distance in feet and inches:" << endl;
    db.getdata();
    add(dm, db);
    return 0;
}
```

ASSIGNMENT 04 (matrix operations)

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

class MAT
{
private:
    vector<vector<int>> matrix;
    int m, n;

public:
    MAT(int m, int n)
    {
        this->m = m;
        this->n = n;
        matrix.resize(m, vector<int>(n, 0));
    }
    void inputMatrix()
    {
        cout << "Enter the elements of the matrix : " << endl;
```

```
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                cin >> matrix[i][j];
            }
        }
    }
    void displayMatrix()
    {
        cout << "Thematrixis:" << endl;
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                cout << matrix[i][j] << " ";
            }
            cout << endl;
        }
    }
    MAT add(MAT &other)
    {
        MAT result(m, n);
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                result.matrix[i][j] = matrix[i][j] + other.matrix[i][j];
            }
        }
        return result;
    }
    MAT subtract(MAT &other)
    {
        MAT result(m, n);
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                result.matrix[i][j] = matrix[i][j] - other.matrix[i][j];
            }
        }
        return result;
    }
    MAT multiply(MAT &other)
    {
        if (n != other.m)
        {
            throw "Invaliddimensionsformatrixmultiplication!";
        }
        MAT result(m, other.n);
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < other.n; j++)
            {
                for (int k = 0; k < n; k++)
                {
                    result.matrix[i][j] += matrix[i][k] * other.matrix[k][j];
                }
            }
        }
        return result;
    }
};
int main()
```

```
{
    int m, n;
    cout << "Enter the number of rows and columns for the matrix:";
    cin >> m >> n;

    MAT mat1(m, n), mat2(m, n);
    cout << "For matrix 1:" << endl;
    mat1.inputMatrix();
    cout << "For matrix 2:" << endl;
    mat2.inputMatrix();

    cout << "Matrix 1 : " << endl;
    mat1.displayMatrix();
    cout << "Matrix 2 : " << endl;
    mat2.displayMatrix();

    MAT mat3 = mat1.add(mat2);
    MAT mat4 = mat1.subtract(mat2);
    MAT mat5 = mat1.multiply(mat2);

    cout << "Result of addition: " << endl;
    mat3.displayMatrix();
    cout << "Result of subtraction: " << endl;
    mat4.displayMatrix();
    cout << "Result of multiplication: " << endl;
    mat5.displayMatrix();
    return 0;
}
```

ASSIGNMENT 05 (Stud class – constructor and destructor)

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

class Stud
{
private:
    std::string name;
    int rollNo;
    int age;

public:

    Stud() : name("none"), rollNo(0), age(0)
    {
        cout << "Default Constructor Called" << endl;
    }

    Stud(string n, int r, int a) : name(n), rollNo(r), age(a)
    {
        cout << "\nMultiple Constructor Called" << endl;
    }

    Stud(const Stud &other) : name(other.name),rollNo(other.rollNo), age(other.age)
    {
        cout << "\nCopy Constructor Called" << endl;
    }

    Stud(string n, int r) : name(n), rollNo(r), age(0)
    {
        cout << "\nOverloaded Constructor Called" << endl;
    }
```

```
    ~Stud()
    {
        cout << "\nDestructor Called for " << name << endl;
    }
    void displayInfo()
    {
        cout << "Name: " << name << ", Roll No: " << rollNo << ", Age: " << age << endl;
    }
};
int main()
{
    Stud student1;
    student1.displayInfo();

    Stud student2("Alice", 101, 20);
    student2.displayInfo();

    Stud student3 = student2;
    student3.displayInfo();

    Stud student4("Bob", 102);
    student4.displayInfo();

    return 0;
}
```

ASSIGNMENT 06 (person,admin, admin, master – multiple inheritance)

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

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

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

    virtual void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

class Account : virtual public Person {
protected:
    string accountID;

public:
    Account(const string& name, int age, const string& accID) : Person(name, age),
accountID(accID) {}

    virtual void displayInfo() override {
        Person::displayInfo();
        cout << "Account ID: " << accountID << endl;
    }
};

class Admin : virtual public Person {
protected:
```

```
    string adminRole;

public:
    Admin(const string& name, int age, const string& role) : Person(name, age), adminRole(role)
{}

    virtual void displayInfo() override {
        Person::displayInfo();
        cout << "Admin Role: " << adminRole << endl;
    }
};

class Master : public Account, public Admin {
public:
    Master(const string& name, int age, const string& accID, const string& role)
        : Person(name, age), Account(name, age, accID), Admin(name, age, role) {}

    void displayInfo() override {
        Person::displayInfo();
        Account::displayInfo();
        Admin::displayInfo();
    }
};

int main() {
    Master master("John Doe", 30, "12345", "Administrator");

    cout << "Information in Master object:" << endl;
    master.displayInfo();

    return 0;
}
```

ASSIGNMENT 07 (books and media – polymorphism)

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

class Media
{
protected:
    string title;
    double price;

public:
    Media(const string &t, double p) : title(t), price(p) {}
    virtual void display()
    {
        cout << "Title: " << title << "\nPrice: Rs." << price << endl;
    }
};
class Book : public Media
{
    int numPages;

public:
    Book(const string &t, double p, int pages) : Media(t, p), numPages(pages) {}
    void display() override
```

```
    {
        cout << "Book Details:\n";
        Media::display();
        cout << "Number of Pages: " << numPages << endl;
    }
};

class VideoTape : public Media
{
    int playTime;

public:
    VideoTape(const std::string &t, double p, int time) : Media(t, p), playTime(time) {}
    void display() override
    {
        cout << "Video Tape Details:\n";
        Media::display();
        cout << "Playing Time: " << playTime << " minutes" << endl;
    }
};

int main()
{
    Media *items[3];
    items[0] = new Book("C++ Programming", 440, 500);
    items[1] = new VideoTape("Introduction to AI", 1200, 120);
    items[2] = new Book("Data Structures in C", 600, 980);
    for (int i = 0; i < 3; i++)
    {
        items[i]->display();
        cout << std::endl;
        delete items[i];
    }
    return 0;
}
```

ASSIGNMRNT 08 (this pointer, new & delete operator)

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

class MyClass {
public:
    MyClass(int value) : data(value) {
    cout << "Object created with value: " << data << endl;
    }

void showValue() {
    cout << "Value of this object: " << this->data << endl;
    }

void updateValue(int newValue) {
    this->data = newValue;
    }

void releaseMemory() {
    delete this;
    }

~MyClass() {
    cout << "Object destroyed with value: " << data << endl;
    }
```

```
private:
    int data;
};


int main() {
MyClass* obj1 = new MyClass(42);
obj1->showValue();
obj1->updateValue(100);
obj1->showValue();
obj1->releaseMemory();
return 0;
}
```

ASSIGNMENT 09 (template function)

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

template <typename T>
T findMinimum(const T arr[], int size)
{
    if (size <= 0)
    {
        cout << "Error: Array is empty or size is invalid." << endl;
        return T();
    }
    T min = arr[0];
    for (int i = 1; i < size; i++)
    {
        if (arr[i] < min)
        {
            min = arr[i];
        }
    }
    return min;
}
int main()
{
    int intArr[] = {5, 2, 8, 1, 7};
    double doubleArr[] = {3.14, 2.71, 1.618, 0.577};
    int intMin = findMinimum(intArr, sizeof(intArr) / sizeof(intArr[0]));
    double doubleMin = findMinimum(doubleArr, sizeof(doubleArr)) / sizeof(doubleArr[0]);
    cout << "Minimum value in the integer array: " << intMin << endl;
    cout << "Minimum value in the double array: " << doubleMin << endl;
    return 0;
}
```

ASSIGNMENT 10 (try & catch – error handling)

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

int main() {
    int numerator, denominator, result;

    cout << "Enter numerator: ";
    cin >> numerator;

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

```
    try {
        if (denominator == 0) {
            throw "Division by zero error not possible";
        }

        result = numerator / denominator;
        cout << "Result of division: " << result << endl;
    } catch (const char* errorMessage) {
        cout << "Error: " << errorMessage << endl;
    } catch (...) {
        cout << "An unexpected error occurred." << endl;
    }

    return 0;
}
```