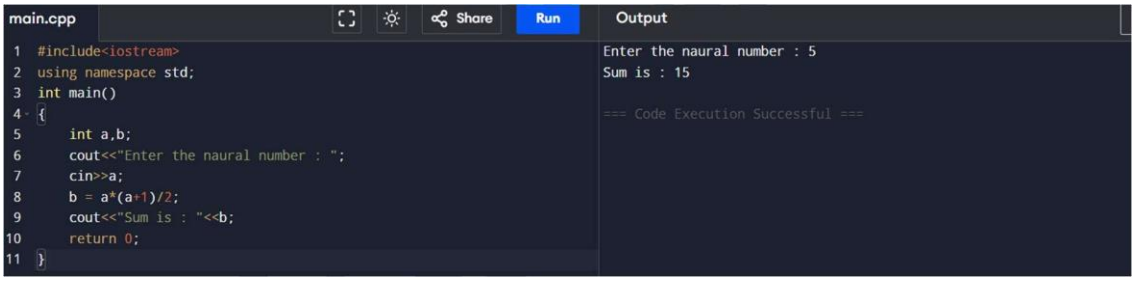
Day-1 Solution

1) Sum of Natural Numbers up to N

Code:-

```
#include<iostream>

using namespace std;

int main()

{

   int a,b;

   cout<<"Enter the naural number : ";

   cin>>a;

   b = a*(a+1)/2;

   cout<<"Sum is : "<<b;

   return 0;

}
```

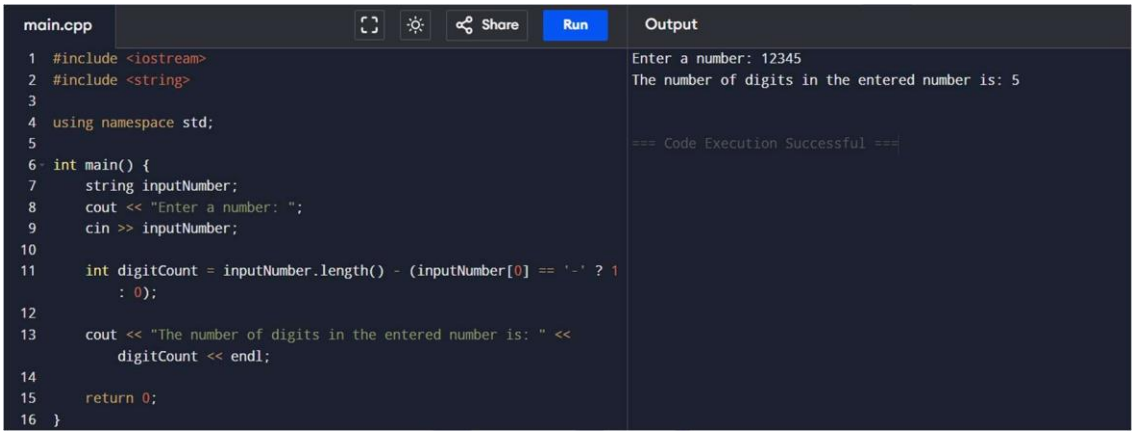


2) Count Digits in a Number

Code:-

```
#include <iostream>

#include <string>


using namespace std;


int main() {

   string inputNumber;

   cout << "Enter a number: ";

   cin >> inputNumber;
```

```
    int digitCount = inputNumber.length() - (inputNumber[0] == '-' ? 1 : 0);

    cout << "The number of digits in the entered number is: " << digitCount << endl;

    return 0;
}
```

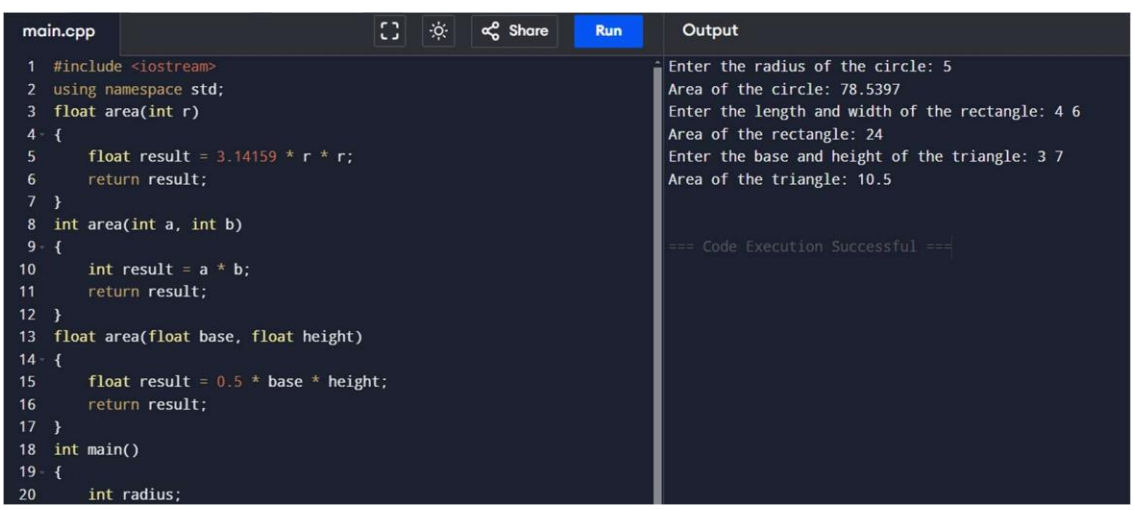


3) Function Overloading for Calculating Area.

Code:-

```
#include <iostream>

using namespace std;

float area(int r)

{

    float result = 3.14159 * r * r;

    return result;

}

int area(int a, int b)

{

    int result = a * b;

    return result;

}

float area(float base, float height)
```

```
{
    float result = 0.5 * base * height;
    return result;
}
int main()
{
    int radius;
    int length, width;
    float base, height;
    cout << "Enter the radius of the circle: ";
    cin >> radius;
    cout << "Area of the circle: " << area(radius) << endl;
    cout << "Enter the length and width of the rectangle: ";
    cin >> length >> width;
    cout << "Area of the rectangle: " << area(length, width) << endl;
    cout << "Enter the base and height of the triangle: ";
    cin >> base >> height;
    cout << "Area of the triangle: " << area(base, height) << endl;5

    return 0;
}
```



4) Implement Polymorphism for Banking Transactions

Code:-

```
#include <iostream>
#include <iomanip>
using namespace std;
class Account {
protected:
   double balance;
public:
   Account(double bal) : balance(bal) {}
   virtual void calculateInterest() = 0;
   virtual ~Account() {}
};
class SavingsAccount : public Account {
   double rate;
   int time;
public:
   SavingsAccount(double bal, double r, int t) : Account(bal), rate(r), time(t) {}
   void calculateInterest() override {
      double interest = balance * (rate / 100) * time;
      cout << "Savings Account Interest: " << fixed << setprecision(2) << interest << endl;
   }
};
class CurrentAccount : public Account {
   double maintenanceFee;
public:
   CurrentAccount(double bal, double fee) : Account(bal), maintenanceFee(fee) {}
   void calculateInterest() override {
      balance -= maintenanceFee;
```

```
        cout << "Current Account Balance after Maintenance Fee: " << fixed << setprecision(2) <<
balance << endl;
    }
};
int main() {
    int accountType;
    double balance;
    cout << "Enter Account Type (1 for Savings, 2 for Current): ";
    cin >> accountType;

    if (accountType < 1 || accountType > 2) {
        cout << "Invalid account type. Please enter 1 or 2." << endl;
        return 1;
    }
    cout << "Enter Account Balance: ";
    cin >> balance;
    if (balance < 1000 || balance > 1000000) {
        cout << "Invalid balance. Please enter a value between 1000 and 1000000." << endl;
        return 1;
    }
    if (accountType == 1) {
        double rate;
        int time;
        cout << "Enter Interest Rate (%): ";
        cin >> rate;
        if (rate < 1 || rate > 15) {
            cout << "Invalid interest rate. Please enter a value between 1 and 15." << endl;
            return 1;
        }
        cout << "Enter Time (in years): ";
        cin >> time;
```

```
        if (time < 1 || time > 10) {
            cout << "Invalid time. Please enter a value between 1 and 10 years." << endl;
            return 1;
        }
        SavingsAccount savings(balance, rate, time);
        savings.calculateInterest();
    } else if (accountType == 2) {
        double fee;
        cout << "Enter Monthly Maintenance Fee: ";
        cin >> fee;
        if (fee < 50 || fee > 500) {
            cout << "Invalid maintenance fee. Please enter a value between 50 and 500." << endl;
            return 1;
        }
        CurrentAccount current(balance, fee);
        current.calculateInterest();
    }
    return 0;
}
```

```
#include <iostream>
#include <iomanip>
using namespace std;
class Account {
protected:
    double balance;
public:
    Account(double bal) : balance(bal) {}
    virtual void calculateInterest() = 0;
    virtual ~Account() {}
};
class SavingsAccount : public Account {
    double rate;
    int time;
public:
    SavingsAccount(double bal, double r, int t) : Account(bal), rate
        (r), time(t) {}
    void calculateInterest() override {
        double interest = balance * (rate / 100) * time;
        cout << "Savings Account Interest: " << fixed <<
            setprecision(2) << interest << endl;
    }
};
class CurrentAccount : public Account {
    double maintenanceFee;
```

```
Enter Account Type (1 for Savings, 2 for Current): 1
Enter Account Balance: 10000
Enter Interest Rate (%): 5
Enter Time (in years): 3
Savings Account Interest: 1500.00


=== Code Execution Successful ===
```

5) Hierarchical Inheritance for Employee Management System

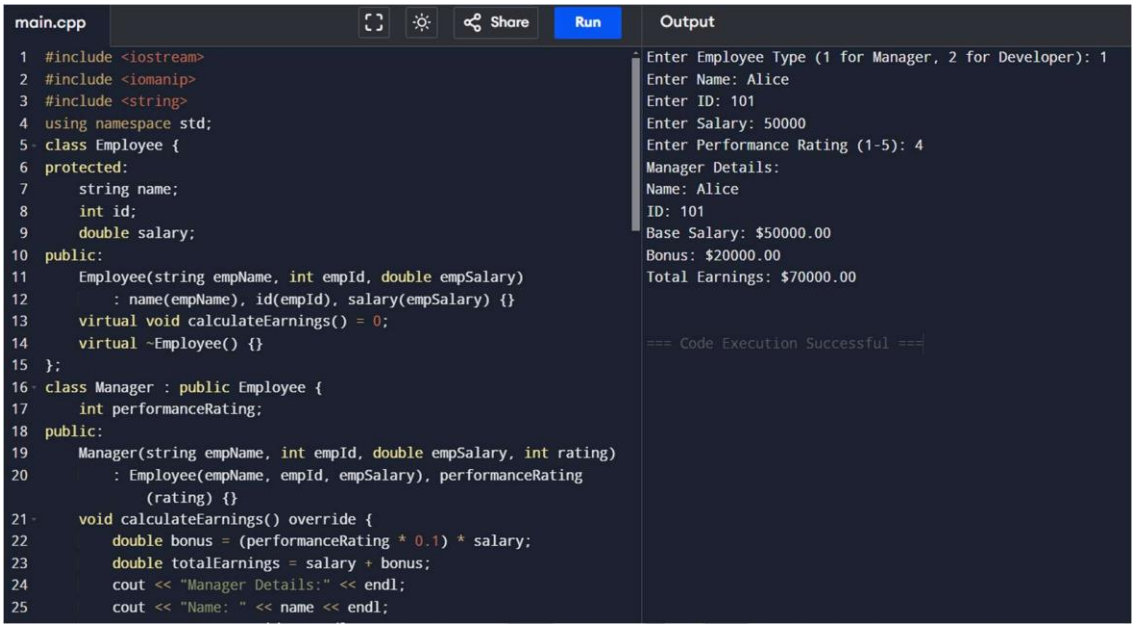
Code:-

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
class Employee {
protected:
   string name;
   int id;
   double salary;
public:
   Employee(string empName, int empId, double empSalary)
      : name(empName), id(empId), salary(empSalary) {}
   virtual void calculateEarnings() = 0;
   virtual ~Employee() {}
};
class Manager : public Employee {
   int performanceRating;
public:
   Manager(string empName, int empId, double empSalary, int rating)
      : Employee(empName, empId, empSalary), performanceRating(rating) {}
   void calculateEarnings() override {
      double bonus = (performanceRating * 0.1) * salary;
      double totalEarnings = salary + bonus;
      cout << "Manager Details:" << endl;
      cout << "Name: " << name << endl;
      cout << "ID: " << id << endl;
      cout << "Base Salary: $" << fixed << setprecision(2) << salary << endl;
```

```
        cout << "Bonus: $" << bonus << endl;
        cout << "Total Earnings: $" << totalEarnings << endl;
    }
};
class Developer : public Employee {
    int extraHours;
public:
    Developer(string empName, int empId, double empSalary, int hours)
        : Employee(empName, empId, empSalary), extraHours(hours) {}
    void calculateEarnings() override {
        double overtime = extraHours * 500;
        double totalEarnings = salary + overtime;
        cout << "Developer Details:" << endl;
        cout << "Name: " << name << endl;
        cout << "ID: " << id << endl;
        cout << "Base Salary: $" << fixed << setprecision(2) << salary << endl;
        cout << "Overtime Compensation: $" << overtime << endl;
        cout << "Total Earnings: $" << totalEarnings << endl;
    }
};
int main() {
    int employeeType;
    string name;
    int id;
    double salary;
    cout << "Enter Employee Type (1 for Manager, 2 for Developer): ";
    cin >> employeeType;
    if (employeeType < 1 || employeeType > 2) {
        cout << "Invalid employee type. Please enter 1 or 2." << endl;
        return 1;
    }
```

```
cout << "Enter Name: ";
cin.ignore(); // To clear the newline character from input buffer
getline(cin, name);
cout << "Enter ID: ";
cin >> id;
cout << "Enter Salary: ";
cin >> salary;
if (salary < 10000 || salary > 1000000) {
    cout << "Invalid salary. Please enter a value between 10,000 and 1,000,000." << endl;
    return 1;
}
if (employeeType == 1) {
    int rating;
    cout << "Enter Performance Rating (1-5): ";
    cin >> rating;
    if (rating < 1 || rating > 5) {
        cout << "Invalid rating. Please enter a value between 1 and 5." << endl;
        return 1;
    }
    Manager manager(name, id, salary, rating);
    manager.calculateEarnings();
} else if (employeeType == 2) {
    int extraHours;
    cout << "Enter Extra Hours Worked: ";
    cin >> extraHours;
    if (extraHours < 0 || extraHours > 100) {
        cout << "Invalid extra hours. Please enter a value between 0 and 100." << endl;
        return 1;
    }
    Developer developer(name, id, salary, extraHours);
    developer.calculateEarnings();
```

```
  }
  return 0;
}
```

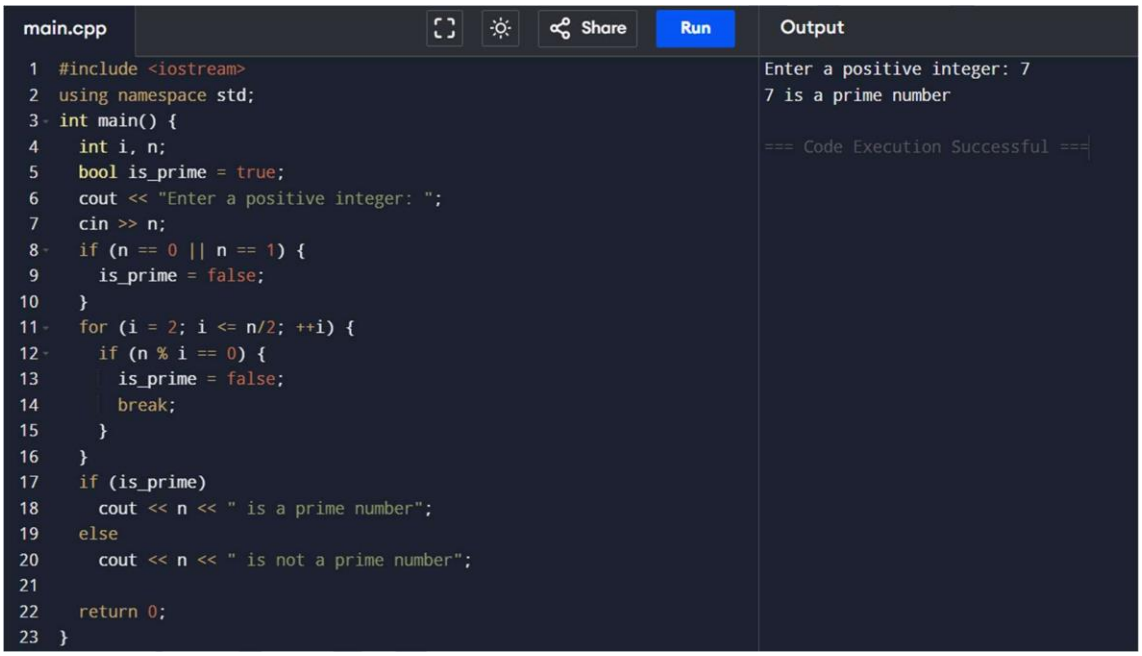


6) Check if a Number is Prime

Code:-

```
#include <iostream>
using namespace std;
int main() {
  int i, n;
  bool is_prime = true;
  cout << "Enter a positive integer: ";
  cin >> n;
  if (n == 0 || n == 1) {
    is_prime = false;
  }
  for (i = 2; i <= n/2; ++i) {
    if (n % i == 0) {
    is_prime = false;
    break;
```

```
    }
  }
  if (is_prime)
    cout << n << " is a prime number";
  else
    cout << n << " is not a prime number";


  return 0;
}
```

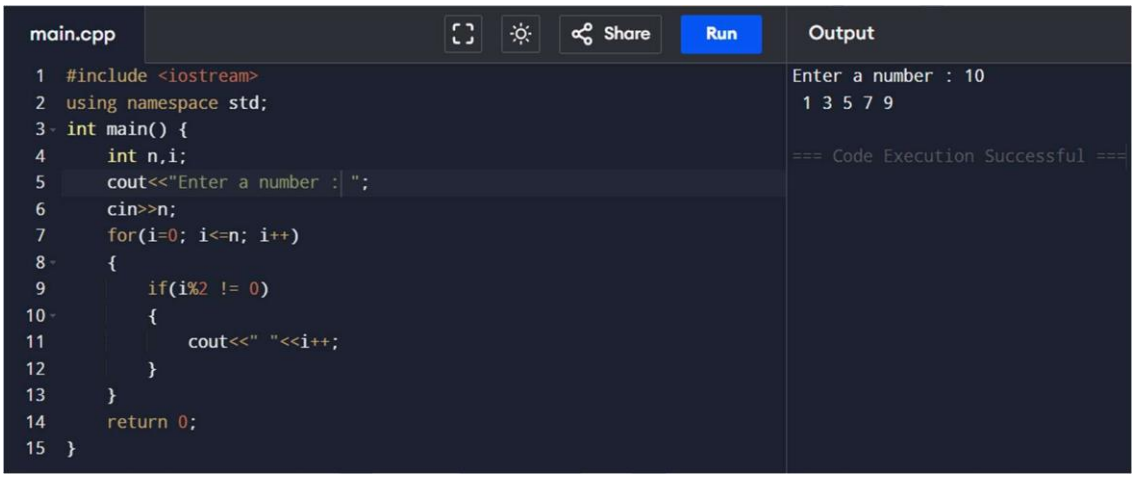


7) Print Odd Numbers up to N

Code:-

```
#include <iostream>
using namespace std;
int main() {
   int n,i;
   cout<<"Enter a number : ";
   cin>>n;
   for(i=0; i<=n; i++)
```

```
    {
        if(i%2 != 0)
        {
            cout<<" "<<i++;
        }
    }
    return 0;
}
```

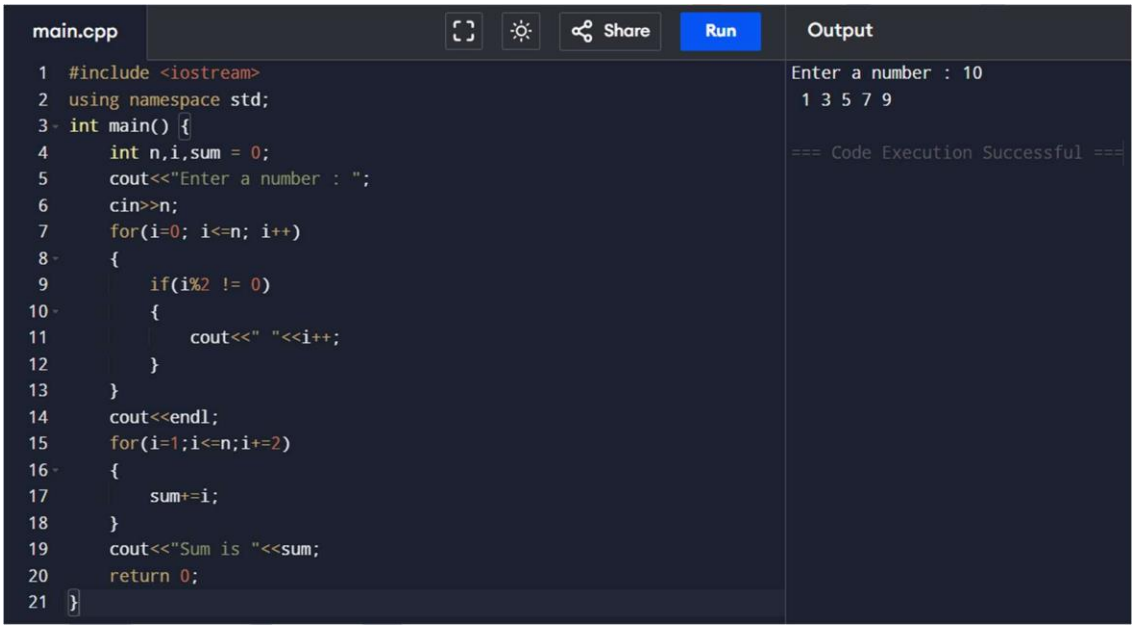


8) Sum of Odd Numbers up to N

Code:-

```
#include <iostream>
using namespace std;
int main() {
    int n,i,sum = 0;
    cout<<"Enter a number : ";
    cin>>n;
    for(i=0; i<=n; i++)
    {
        if(i%2 != 0)
        {
            cout<<" "<<i++;
```

```
    }
  }
  cout<<endl;
  for(i=1;i<=n;i+=2)
  {
    sum+=i;
  }
  cout<<"Sum is "<<sum;
  return 0;
}
```



9) Print Multiplication Table of a Number

Code:-

```
#include <iostream>
using namespace std;
int main()
{
  int a,i;
  cout<<"Enter Number : ";
  cin>>a;
```

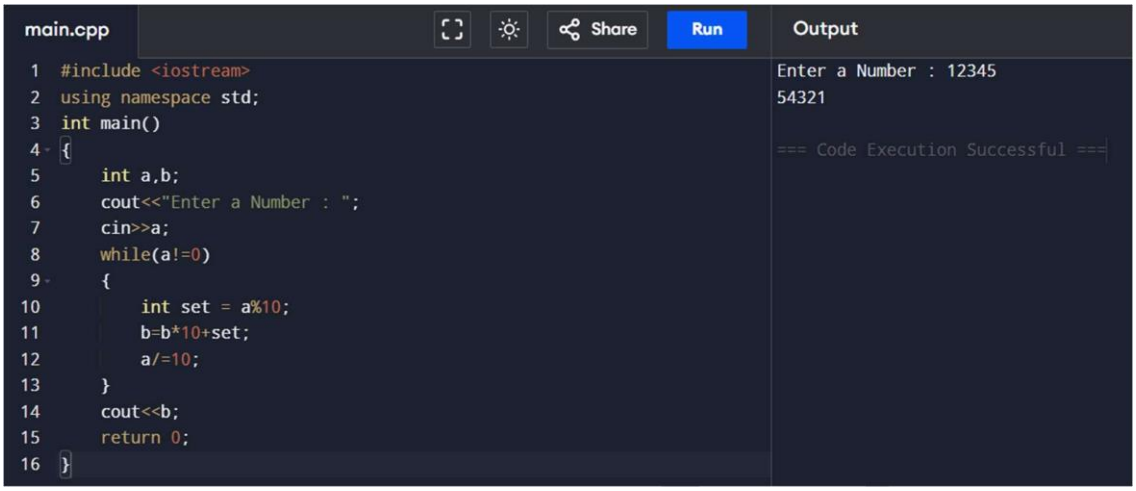
```
    for(i=1;i<=10;i++)
    {
        cout<<a<<" * "<<i<<" = "<<i*a<<endl;
    }
    return 0;
}
```

10) Reverse a Number

Code:-

```
#include <iostream>
using namespace std;
int main()
{
    int a,b;
    cout<<"Enter a Number : ";
    cin>>a;
    while(a!=0)
    {
        int set = a%10;
        b=b*10+set;
        a/=10;
    }
    cout<<b;
```

```
    return 0;
}
```

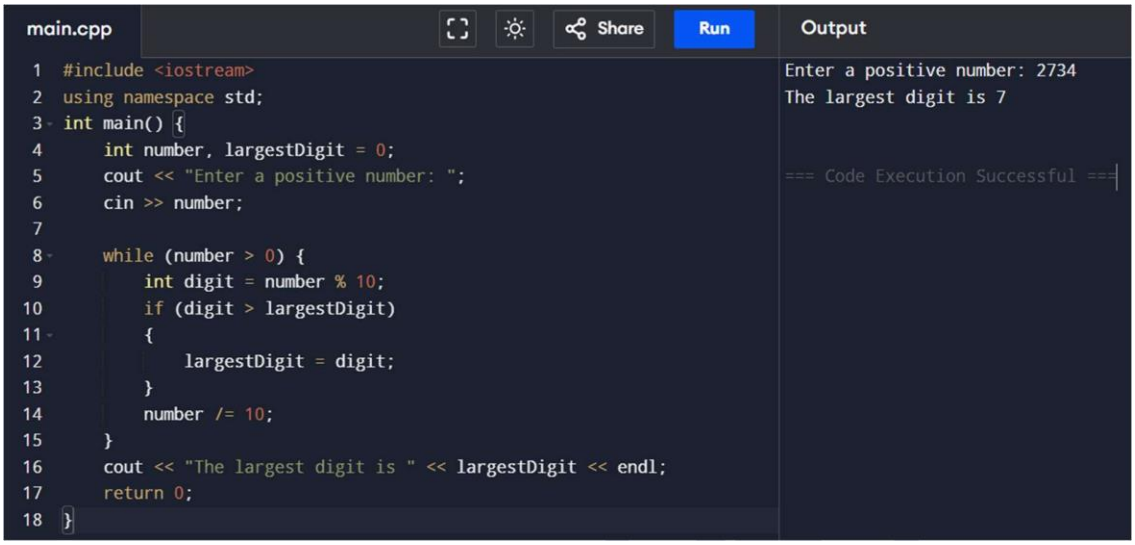


11) Find the Largest Digit in a Number

Code:-

```
#include <iostream>
using namespace std;
int main() {
    int number, largestDigit = 0;
    cout << "Enter a positive number: ";
    cin >> number;

    while (number > 0) {
        int digit = number % 10;
        if (digit > largestDigit)
        {
            largestDigit = digit;
        }
        number /= 10;
    }
    cout << "The largest digit is " << largestDigit << endl;
    return 0;
```

}



12) Check if a Number is a Palindrome

Code:-

```
#include <iostream>

using namespace std;

int main()

{

   int a,b,c;

   cout<<"Enter a Number : ";

   cin>>a;

   int n=a;

   while(a!=0)

   {

      int set = a%10;

      b=b*10+set;

      a/=10;

   }

   cout<<b;

   if(n==b)
```

```
  {
     cout<<" Is Palindrome ";
  }
  else
  cout<<" Not Palindrome ";
  return 0;
}
```

main.cpp

```
#include <iostream>
using namespace std;
int main()
{
    int a,b,c;
    cout<<"Enter a Number : ";
    cin>>a;
    int n=a;
    while(a!=0)
    {
        int set = a%10;
        b=b*10+set;
        a/=10;
    }
    cout<<b;
    if(n==b)
    {
        cout<<" Is Palindrome ";
    }
    else
    cout<<" Not Palindrome ";
    return 0;
}
```

Output

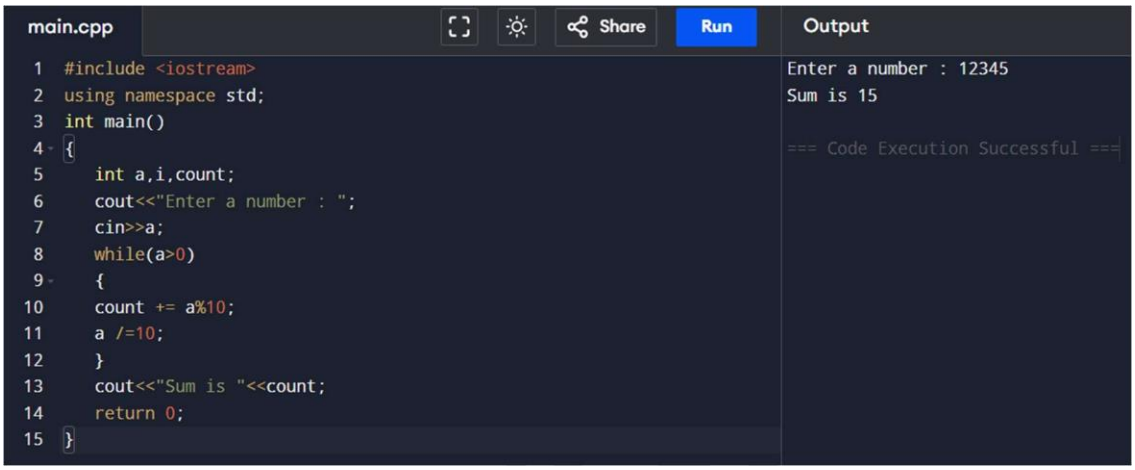
```
Enter a Number : 121
121 Is Palindrome

=== Code Execution Successful ===
```

13) Find the Sum of Digits of a Number

Code:-

```
#include <iostream>
using namespace std;
int main()
{
  int a,i,count;
  cout<<"Enter a number : ";
  cin>>a;
  while(a>0)
  {
```

```
  count += a%10;
  a /=10;
  }
  cout<<"Sum is "<<count;
  return 0;
}
```

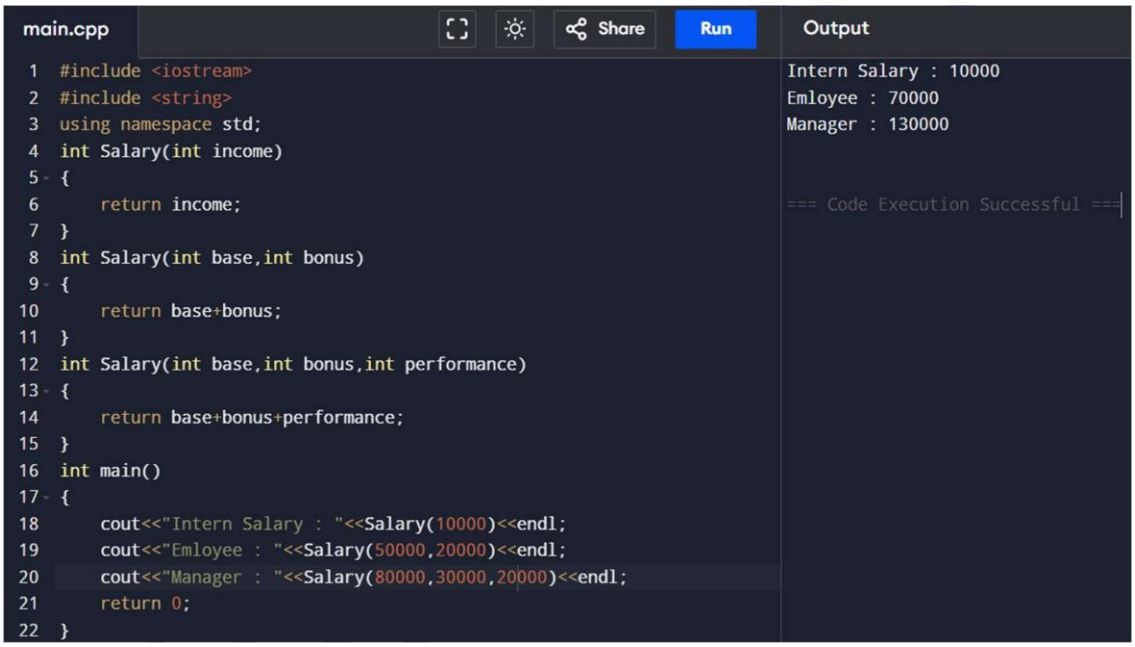


14) Function Overloading with Hierarchical Structure.

Code:-

```
#include <iostream>
#include <string>
using namespace std;
int Salary(int income)
{
   return income;
}
int Salary(int base,int bonus)
{
   return base+bonus;
}
int Salary(int base,int bonus,int performance)
{
```

```
    return base+bonus+performance;
}
int main()
{
    cout<<"Intern Salary : "<<Salary(10000)<<endl;
    cout<<"Emloyee : "<<Salary(50000,20000)<<endl;
    cout<<"Manager : "<<Salary(80000,30000,20000)<<endl;
    return 0;
}
```



15) Encapsulation with Employee Details

Code:-

```
#include <iostream>
#include <string>
using namespace std;
class Employee
{
    private:
    int EID;
```

```
    string EName;
    float Salary;
    friend void detail();
};
void detail()
{
    Employee a1;
    cout<<"Enter Employee ID : ";
    cin>>a1.EID;
    cin.ignore();
    cout<<"Enter Employee Name : ";
    getline(cin, a1.EName);
    cout<<"Enter Salary : ";
    cin>>a1.Salary;
    cout<<"Employee ID : "<<a1.EID<<endl;
    cout<<"Employee Name : "<<a1.EName<<endl;
    cout<<"Employee Salary : "<<a1.Salary<<endl;
}
int main()
{
    detail();
    return 0;
}
```

main.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
class Employee
{
    private:
    int EID;
    string EName;
    float Salary;
    friend void detail();
};
void detail()
{
    Employee a1;
    cout<<"Enter Employee ID : ";
    cin>>a1.EID;
    cin.ignore();
    cout<<"Enter Employee Name : ";
    getline(cin, a1.EName);
    cout<<"Enter Salary : ";
    cin>>a1.Salary;
    cout<<"Employee ID : "<<a1.EID<<endl;
    cout<<"Employee Name : "<<a1.EName<<endl;
    cout<<"Employee Salary : "<<a1.Salary<<endl;
```

Output

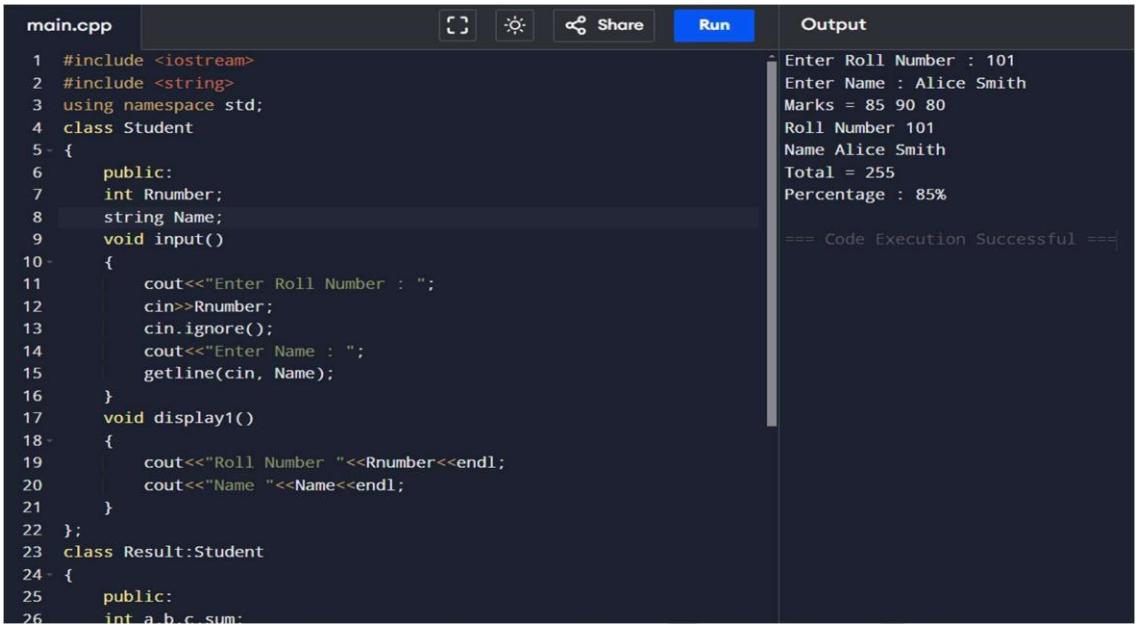
```
Enter Employee ID : 101
Enter Employee Name : John Doe
Enter Salary : 75000.5
Employee ID : 101
Employee Name : John Doe
Employee Salary : 75000.5


=== Code Execution Successful ===
```

16) Inheritance with Student and Result Classes.

Code:-

```cpp
#include <iostream>

#include <string>

using namespace std;

class Student

{

   public:

   int Rnumber;

   string Name;

   void input()

   {

      cout<<"Enter Roll Number : ";

      cin>>Rnumber;

      cin.ignore();

      cout<<"Enter Name : ";

      getline(cin, Name);

   }

   void display1()

   {

      cout<<"Roll Number "<<Rnumber<<endl;
```

```
        cout<<"Name  "<<Name<<endl;
    }
};
class Result:Student
{
    public:
    int a,b,c,sum;
    float P;
    void Marks()
    {
        cout<<"Marks = ";
        cin>>a>>b>>c;
        sum = a+b+c;
        P=(float)sum/300*100;
    }
    void display2()
    {
        cout<<"Total = "<<sum<<endl;
        cout<<"Percentage : "<<P<<"%";
    }
};
int main()
{
    Student a1;
    Result a2;
    a1.input();
    a2.Marks();
    a1.display1();
    a2.display2();
}
```



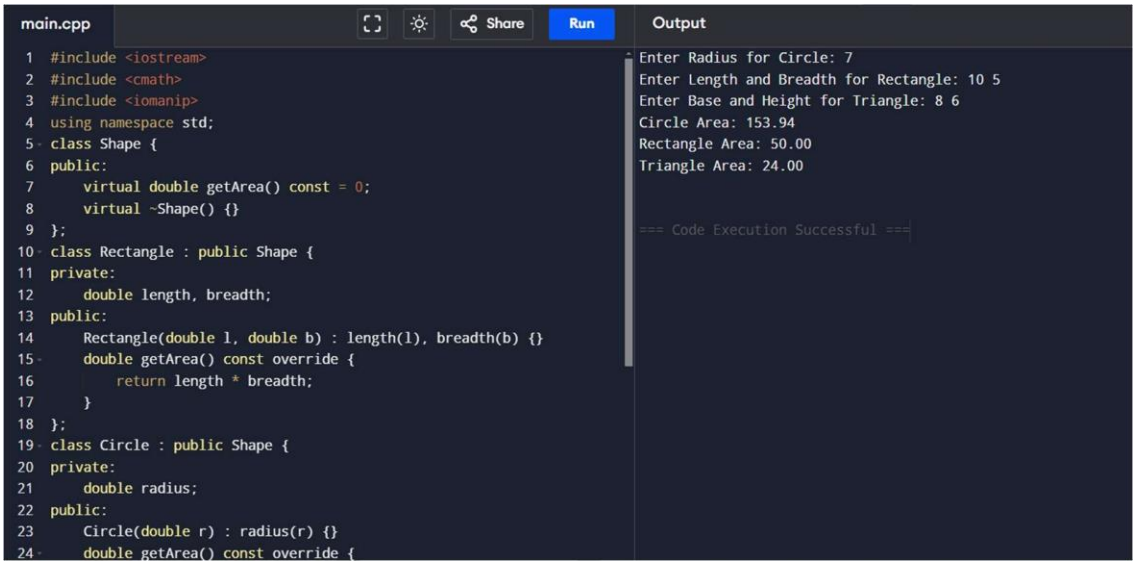
17) Polymorphism with Shape Area Calculation.

Code:-

```
#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

class Shape {

public:

   virtual double getArea() const = 0;

   virtual ~Shape() {}

};

class Rectangle : public Shape {

private:

   double length, breadth;

public:

   Rectangle(double l, double b) : length(l), breadth(b) {}

   double getArea() const override {

      return length * breadth;

   }

};
```

```
class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double getArea() const override {
        return 3.14159 * radius * radius;
    }
};
class Triangle : public Shape {
private:
    double base, height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
    double getArea() const override {
        return 0.5 * base * height;
    }
};
int main() {
    double radius, length, breadth, base, height;
    cout << "Enter Radius for Circle: ";
    cin >> radius;
    cout << "Enter Length and Breadth for Rectangle: ";
    cin >> length >> breadth;
    cout << "Enter Base and Height for Triangle: ";
    cin >> base >> height;
    Shape* circle = new Circle(radius);
    Shape* rectangle = new Rectangle(length, breadth);
    Shape* triangle = new Triangle(base, height);
    cout << fixed << setprecision(2);
    cout << "Circle Area: " << circle->getArea() << endl;
```

```
    cout << "Rectangle Area: " << rectangle->getArea() << endl;
    cout << "Triangle Area: " << triangle->getArea() << endl;
    delete circle;
    delete rectangle;
    delete triangle;
    return 0;
}
```



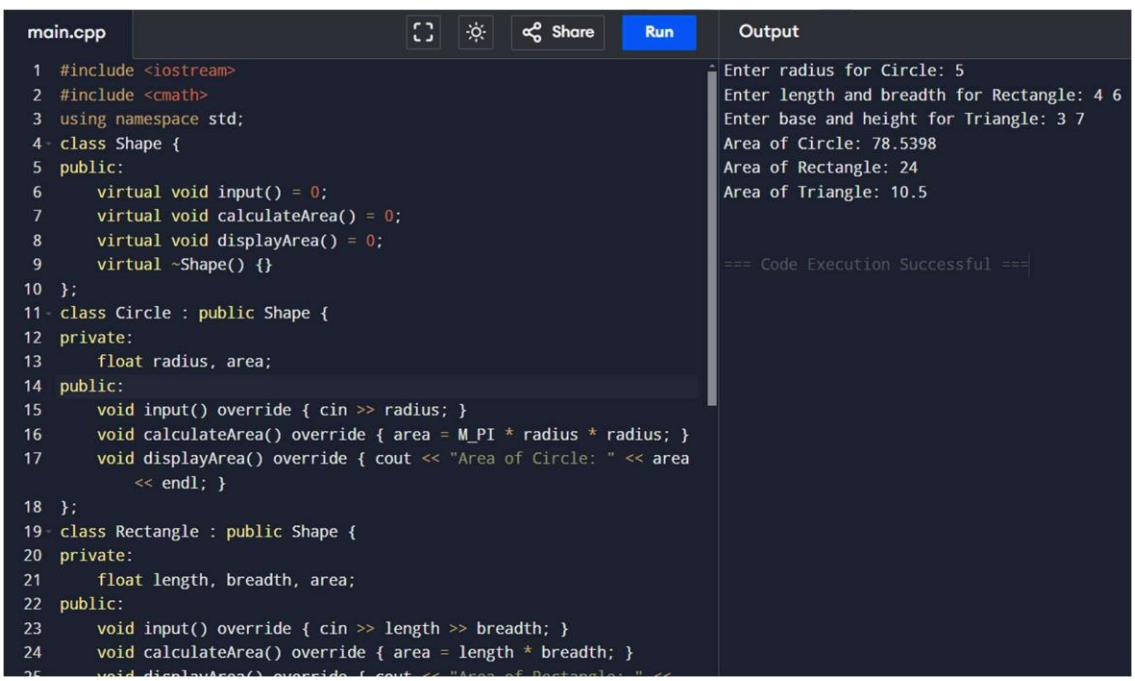
18) Implementing Polymorphism for Shape Hierarchies.

Code:-

```
#include <iostream>
#include <cmath>
using namespace std;
class Shape {
public:
    virtual void input() = 0;
    virtual void calculateArea() = 0;
    virtual void displayArea() = 0;
    virtual ~Shape() {}
};
class Circle : public Shape {
```

```
private:
    float radius, area;
public:
    void input() override { cin >> radius; }
    void calculateArea() override { area = M_PI * radius * radius; }
    void displayArea() override { cout << "Area of Circle: " << area << endl; }
};
class Rectangle : public Shape {
private:
    float length, breadth, area;
public:
    void input() override { cin >> length >> breadth; }
    void calculateArea() override { area = length * breadth; }
    void displayArea() override { cout << "Area of Rectangle: " << area << endl; }
};
class Triangle : public Shape {
private:
    float base, height, area;
public:
    void input() override { cin >> base >> height; }
    void calculateArea() override { area = 0.5 * base * height; }
    void displayArea() override { cout << "Area of Triangle: " << area << endl; }
};

int main() {
    Shape* shapes[3];
    shapes[0] = new Circle();
    shapes[1] = new Rectangle();
    shapes[2] = new Triangle();
    cout << "Enter radius for Circle: ";
    shapes[0]->input();
```

```
    cout << "Enter length and breadth for Rectangle: ";
    shapes[1]->input();
    cout << "Enter base and height for Triangle: ";
    shapes[2]->input();
    for (int i = 0; i < 3; ++i) {
        shapes[i]->calculateArea();
        shapes[i]->displayArea();
    }
    for (int i = 0; i < 3; ++i) {
        delete shapes[i];
    }
    return 0;
}
```



19) Matrix Multiplication Using Function Overloading

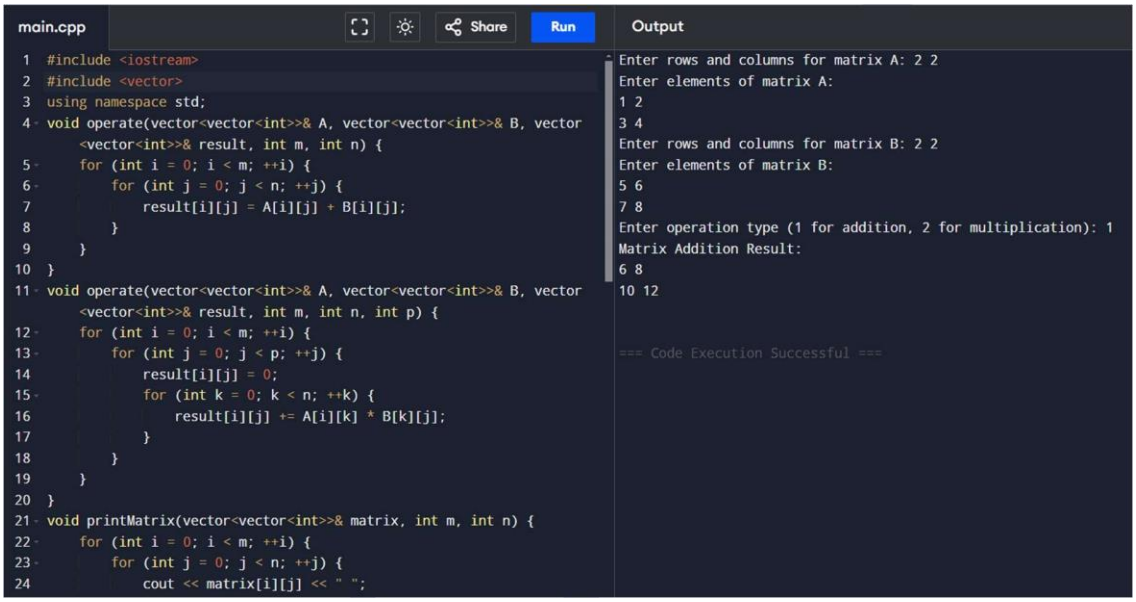
Code:-

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

```
void operate(vector<vector<int>>& A, vector<vector<int>>& B, vector<vector<int>>& result, int m,
int n) {
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            result[i][j] = A[i][j] + B[i][j];
        }
    }
}
void operate(vector<vector<int>>& A, vector<vector<int>>& B, vector<vector<int>>& result, int m,
int n, int p) {
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < p; ++j) {
            result[i][j] = 0;
            for (int k = 0; k < n; ++k) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
void printMatrix(vector<vector<int>>& matrix, int m, int n) {
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}
int main() {
    int m, n, p;
    int operation;
    cout << "Enter rows and columns for matrix A: ";
    cin >> m >> n;
```

```
vector<vector<int>> A(m, vector<int>(n));
cout << "Enter elements of matrix A:" << endl;
for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
        cin >> A[i][j];
cout << "Enter rows and columns for matrix B: ";
cin >> n >> p;
vector<vector<int>> B(n, vector<int>(p));
cout << "Enter elements of matrix B:" << endl;
for (int i = 0; i < n; ++i)
    for (int j = 0; j < p; ++j)
        cin >> B[i][j];
vector<vector<int>> result(m, vector<int>(p, 0));
cout << "Enter operation type (1 for addition, 2 for multiplication): ";
cin >> operation;
if (operation == 1) {
    if (A.size() == B.size() && A[0].size() == B[0].size()) {
        operate(A, B, result, m, n);
        cout << "Matrix Addition Result:" << endl;
        printMatrix(result, m, n);
    } else {
        cout << "Matrix dimensions must be the same for addition!" << endl;
    }
}
else if (operation == 2) {
    if (A[0].size() == B.size()) {
        operate(A, B, result, m, n, p);
        cout << "Matrix Multiplication Result:" << endl;
        printMatrix(result, m, p);
    } else {
        cout << "Matrix dimensions do not match for multiplication!" << endl;
```

```
        }
    }
    else {
        cout << "Invalid operation type!" << endl;
    }
    return 0;
}
```



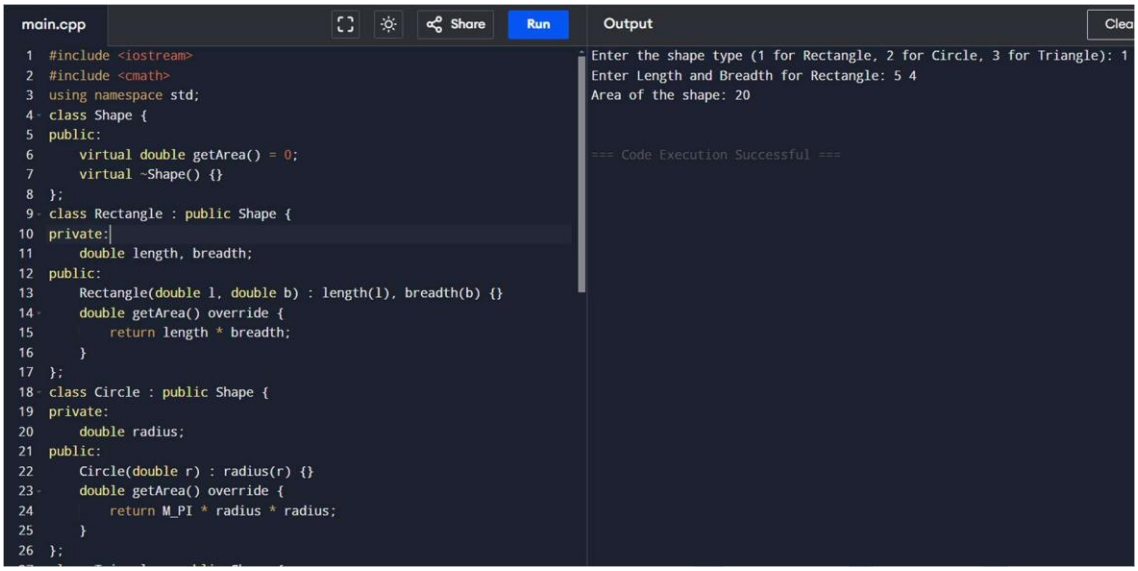
20) Polymorphism in Shape Classes

Code:-

```
#include <iostream>
#include <cmath>
using namespace std;
class Shape {
public:
    virtual double getArea() = 0;
    virtual ~Shape() {}
};
class Rectangle : public Shape {
private:
```

```
    double length, breadth;
public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
    double getArea() override {
        return length * breadth;
    }
};
class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double getArea() override {
        return M_PI * radius * radius;
    }
};
class Triangle : public Shape {
private:
    double base, height;
public:
    Triangle(double b, double h) : base(b), height(h) {}
    double getArea() override {
        return 0.5 * base * height;
    }
};
int main() {
    int choice;
    cout << "Enter the shape type (1 for Rectangle, 2 for Circle, 3 for Triangle): ";
    cin >> choice;
    Shape* shape = nullptr;
    if (choice == 1) {
```

```
        double length, breadth;
        cout << "Enter Length and Breadth for Rectangle: ";
        cin >> length >> breadth;
        shape = new Rectangle(length, breadth);
    }
    else if (choice == 2) {
        double radius;
        cout << "Enter Radius for Circle: ";
        cin >> radius;
        shape = new Circle(radius);
    }
    else if (choice == 3) {
        double base, height;
        cout << "Enter Base and Height for Triangle: ";
        cin >> base >> height;
        shape = new Triangle(base, height);
    }
    else {
        cout << "Invalid choice!" << endl;
        return 0;
    }
    cout << "Area of the shape: " << shape->getArea() << endl;
    delete shape;
    return 0;
}
```



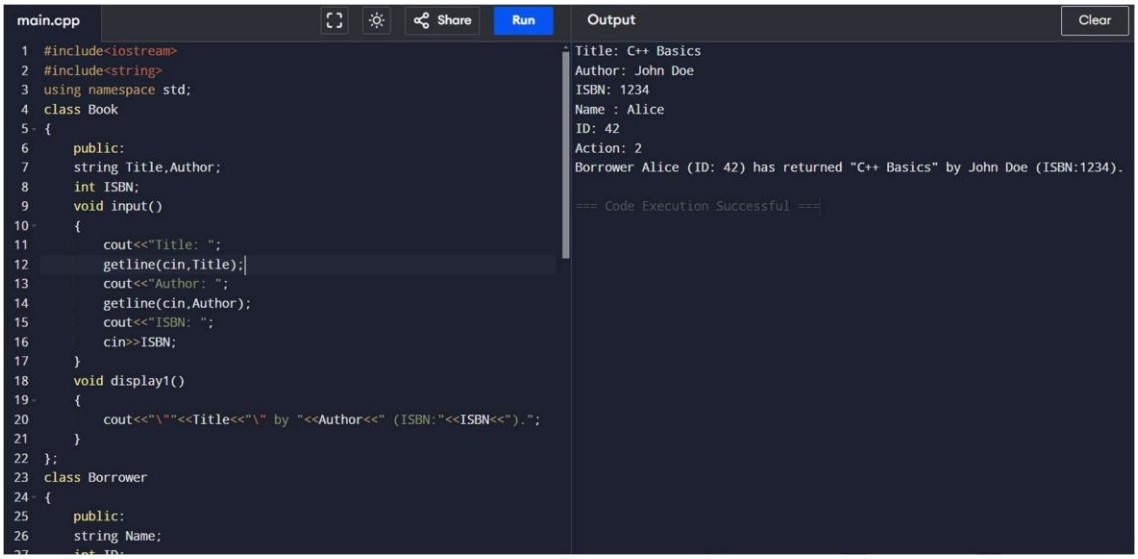
21) Implement Multiple Inheritance to Simulate a Library System

Code:-

```
#include<iostream>

#include<string>

using namespace std;

class Book

{

   public:

   string Title,Author;

   int ISBN;

   void input()

   {

      cout<<"Title: ";

      getline(cin,Title);

      cout<<"Author: ";

      getline(cin,Author);

      cout<<"ISBN: ";

      cin>>ISBN;
```

```
    }
    void display1()
    {
        cout<<"\""<<Title<<"\" by "<<Author<<" (ISBN:"<<ISBN<<").";
    }
};
class Borrower
{
    public:
    string Name;
    int ID;
    void detail()
    {
        cout<<"Name : ";
        cin>>Name;
        cout<<"ID: ";
        cin>>ID;
    }
    void display2()
    {
        cout<<"Borrower "<<Name<<" (ID: "<<ID<<") has ";
    }
};
class Library:Book,Borrower
{
    public:
    int a;
    void action()
    {
        cout<<"Action: ";
        cin>>a;
```

```
    }
    void display3()
    {
        if(a==1)
        cout<<"borrowed ";
        else if(a==2)
        cout<<"returned ";
        else
        cout<<"Wrong ";
    }
};
int main()
{
    Book b1;
    Borrower b2;
    Library a1;
    b1.input();
    b2.detail();
    a1.action();
    b2.display2();
    a1.display3();
    b1.display1();
    return 0;
}
```



22) Multi-Level Inheritance for Vehicle Simulation

Code:-

```
#include <iostream>

#include <string>

using namespace std;

class Vehicle {

public:

   string Brand, Model;

   double Mileage;

   void input() {

      cout << "Brand: ";

      cin >> Brand;

      cout << "Model: ";

      cin >> Model;

      cout << "Mileage: ";

      cin >> Mileage;

   }

   void output() {

      cout << "Vehicle: " << Brand << " " << Model << endl;

      cout << "Mileage: " << Mileage << endl;
```

```
    }
};
class Car : public Vehicle {
public:
    double Fuel, Distance;
    void inputCarDetails() {
        cout << "Fuel (gallons): ";
        cin >> Fuel;
        cout << "Distance Covered (miles): ";
        cin >> Distance;
    }
    void calculateFuelEfficiency() {
        double fuelEfficiency = Distance / Fuel;
        cout << "Fuel Efficiency: " << fuelEfficiency << " miles/gallon" << endl;
    }
};
class ElectricCar : public Car {
public:
    double BatteryCapacity, Efficiency;
    void inputElectricCarDetails() {
        cout << "Battery Capacity (kWh): ";
        cin >> BatteryCapacity;
        cout << "Efficiency (miles per kWh): ";
        cin >> Efficiency;
    }
    void calculateRange() {
        double range = BatteryCapacity * Efficiency;
        cout << "Range: " << range << " miles" << endl;
    }
};
int main() {
```

```
    int vehicleType;
    cout << "Enter Vehicle Type (1 for Car, 2 for Electric Car): ";
    cin >> vehicleType;
    if (vehicleType == 1) {
        Car car;
        car.input();
        car.inputCarDetails();
        car.output();
        car.calculateFuelEfficiency();
    }
    else if (vehicleType == 2) {
        ElectricCar eCar;
        eCar.input();
        eCar.inputElectricCarDetails();
        eCar.output();
        eCar.calculateRange();
    }
    else {
        cout << "Invalid vehicle type." << endl;
    }
    return 0;
}
```

main.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;
class Vehicle {
public:
    string Brand, Model;
    double Mileage;
    void input() {
        cout << "Brand: ";
        cin >> Brand;
        cout << "Model: ";
        cin >> Model;
        cout << "Mileage: ";
        cin >> Mileage;
    }
    void output() {
        cout << "Vehicle: " << Brand << " " << Model << endl;
        cout << "Mileage: " << Mileage << endl;
    }
};
class Car : public Vehicle {
public:
    double Fuel, Distance;
    void inputCarDetails() {
        cout << "Fuel (gallons): ";
        cin >> Fuel;
```

Output

```
Enter Vehicle Type (1 for Car, 2 for Electric Car): 1
Brand: Toyota
Model: Corolla
Mileage: 30000
Fuel (gallons): 15
Distance Covered (miles): 300
Vehicle: Toyota Corolla
Mileage: 30000
Fuel Efficiency: 20 miles/gallon


=== Code Execution Successful ===
```

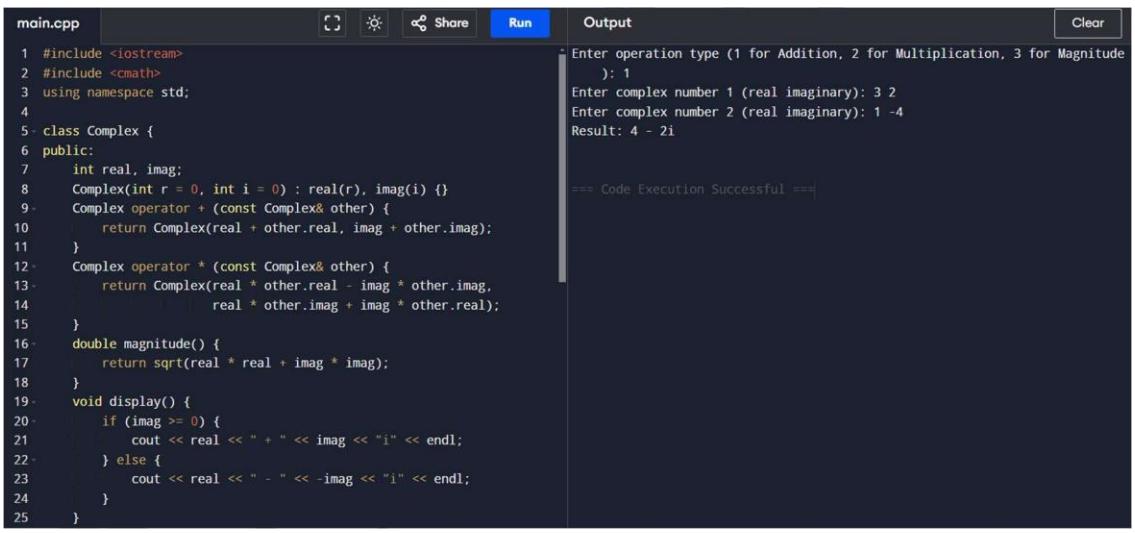
23) Function Overloading for Complex Number Operations.

Code:-

```cpp
#include <iostream>

#include <cmath>

using namespace std;

class Complex {

public:

   int real, imag;

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

   Complex operator + (const Complex& other) {

      return Complex(real + other.real, imag + other.imag);

   }

   Complex operator * (const Complex& other) {

      return Complex(real * other.real - imag * other.imag,

                     real * other.imag + imag * other.real);

   }

   double magnitude() {

      return sqrt(real * real + imag * imag);

   }
```

```
    void display() {
        if (imag >= 0) {
            cout << real << " + " << imag << "i" << endl;
        } else {
            cout << real << " - " << -imag << "i" << endl;
        }
    }
};
int main() {
    int operation;
    cout << "Enter operation type (1 for Addition, 2 for Multiplication, 3 for Magnitude): ";
    cin >> operation;
    if (operation == 1 || operation == 2) {
        int real1, imag1, real2, imag2;
        cout << "Enter complex number 1 (real imaginary): ";
        cin >> real1 >> imag1;
        cout << "Enter complex number 2 (real imaginary): ";
        cin >> real2 >> imag2;
        Complex c1(real1, imag1), c2(real2, imag2), result(0, 0);
        if (operation == 1) {
            result = c1 + c2;
            cout << "Result: ";
            result.display();
        }
        else if (operation == 2) {
            result = c1 * c2;
            cout << "Result: ";
            result.display();
        }
    }
    else if (operation == 3) {
```

```
        int real, imag;
        cout << "Enter complex number (real imaginary): ";
        cin >> real >> imag;
        Complex c(real, imag);
        double mag = c.magnitude();
        cout << "Result: Magnitude = " << mag << endl;
    }
    else {
        cout << "Invalid operation type." << endl;
    }
    return 0;
}
```



24) Area Calculation using Polymorphism

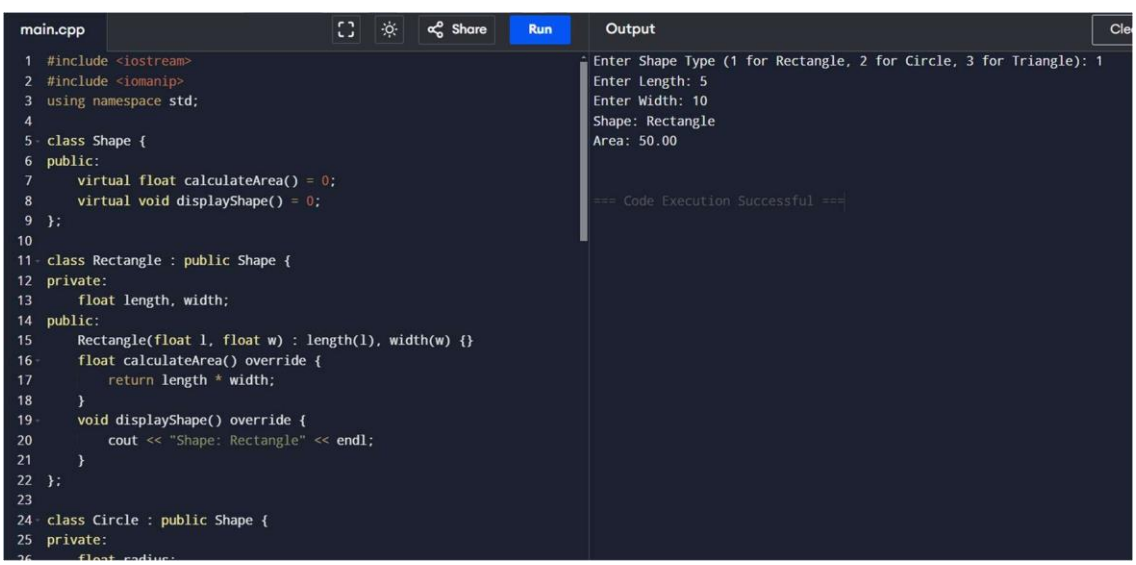
Code:-

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

class Shape {
public:
```

```
    virtual float calculateArea() = 0;
    virtual void displayShape() = 0;
};

class Rectangle : public Shape {
private:
    float length, width;
public:
    Rectangle(float l, float w) : length(l), width(w) {}
    float calculateArea() override {
        return length * width;
    }
    void displayShape() override {
        cout << "Shape: Rectangle" << endl;
    }
};

class Circle : public Shape {
private:
    float radius;
public:
    Circle(float r) : radius(r) {}
    float calculateArea() override {
        return 3.14159 * radius * radius;
    }
    void displayShape() override {
        cout << "Shape: Circle" << endl;
    }
};

class Triangle : public Shape {
```

```
private:
    float base, height;
public:
    Triangle(float b, float h) : base(b), height(h) {}
    float calculateArea() override {
        return 0.5 * base * height;
    }
    void displayShape() override {
        cout << "Shape: Triangle" << endl;
    }
};

int main() {
    int shapeType;
    cout << "Enter Shape Type (1 for Rectangle, 2 for Circle, 3 for Triangle): ";
    cin >> shapeType;

    Shape* shape = nullptr;

    if (shapeType == 1) {
        float length, width;
        cout << "Enter Length: ";
        cin >> length;
        cout << "Enter Width: ";
        cin >> width;
        shape = new Rectangle(length, width);
    } else if (shapeType == 2) {
        float radius;
        cout << "Enter Radius: ";
        cin >> radius;
        shape = new Circle(radius);
```

```
    } else if (shapeType == 3) {
        float base, height;
        cout << "Enter Base: ";
        cin >> base;
        cout << "Enter Height: ";
        cin >> height;
        shape = new Triangle(base, height);
    } else {
        cout << "Invalid shape type." << endl;
        return 0;
    }

    shape->displayShape();
    cout << "Area: " << fixed << setprecision(2) << shape->calculateArea() << endl;

    delete shape;
    return 0;
}
```



25) Advanced Function Overloading for Geometric Shapes

Code:-

```
#include <iostream>
```

```
#include <iomanip>
using namespace std;
float calculateArea(float radius) {
    return 3.14159 * radius * radius;
}
float calculateArea(float length, float breadth) {
    return length * breadth;
}
float calculateArea(double base, double height) {
    return 0.5 * base * height;
}
int main() {
    int choice;
    cout << "Enter Shape Type (1 for Circle, 2 for Rectangle, 3 for Triangle): ";
    cin >> choice;
    if (choice == 1) {
        float radius;
        cout << "Enter Radius: ";
        cin >> radius;
        if (radius <= 0) {
            cout << "Invalid radius. It must be greater than 0." << endl;
            return 0;
        }
        cout << "Shape: Circle" << endl;
        cout << "Radius: " << fixed << setprecision(1) << radius << endl;
        cout << "Area: " << fixed << setprecision(3) << calculateArea(radius) << endl;
    } else if (choice == 2) {
        float length, breadth;
        cout << "Enter Length: ";
        cin >> length;
        cout << "Enter Breadth: ";
```

```
        cin >> breadth;
        if (length <= 0 || breadth <= 0) {
            cout << "Invalid dimensions. Length and breadth must be greater than 0." << endl;
            return 0;
        }
        cout << "Shape: Rectangle" << endl;
        cout << "Length: " << fixed << setprecision(1) << length << endl;
        cout << "Breadth: " << fixed << setprecision(1) << breadth << endl;
        cout << "Area: " << fixed << setprecision(3) << calculateArea(length, breadth) << endl;
    } else if (choice == 3) {
        double base, height;
        cout << "Enter Base: ";
        cin >> base;
        cout << "Enter Height: ";
        cin >> height;
        if (base <= 0 || height <= 0) {
            cout << "Invalid dimensions. Base and height must be greater than 0." << endl;
            return 0;
        }
        cout << "Shape: Triangle" << endl;
        cout << "Base: " << fixed << setprecision(1) << base << endl;
        cout << "Height: " << fixed << setprecision(1) << height << endl;
        cout << "Area: " << fixed << setprecision(3) << calculateArea(base, height) << endl;
    } else {
        cout << "Invalid shape type." << endl;
    }
    return 0;
}
```

main.cpp

Share Run

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

float calculateArea(float radius) {
    return 3.14159 * radius * radius;
}

float calculateArea(float length, float breadth) {
    return length * breadth;
}

float calculateArea(double base, double height) {
    return 0.5 * base * height;
}

int main() {
    int choice;
    cout << "Enter Shape Type (1 for Circle, 2 for Rectangle, 3 for
        Triangle): ";
    cin >> choice;

    if (choice == 1) {
        float radius;
        cout << "Enter Radius: ";
```

Output

Clear

```
Enter Shape Type (1 for Circle, 2 for Rectangle, 3 for Triangle): 1
Enter Radius: 7.5
Shape: Circle
Radius: 7.5
Area: 176.714


=== Code Execution Successful ===
```