

## Day-1 Solution

### 1) Sum of Natural Numbers up to N

**Code:-**

```
#include<iostream>

using namespace std;

int main()
{
    int a,b;

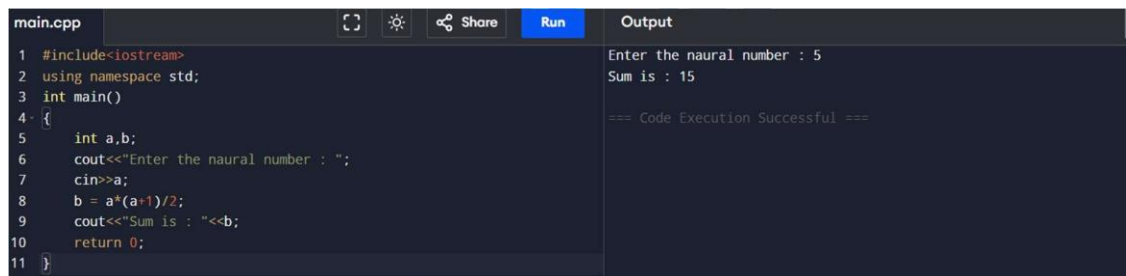
    cout<<"Enter the naural number : ";

    cin>>a;

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

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

    return 0;
}
```



The screenshot shows a C++ IDE with a file named 'main.cpp'. The code is as follows:

```
1 #include<iostream>
2 using namespace std;
3 int main()
4 {
5     int a,b;
6     cout<<"Enter the naural number : ";
7     cin>>a;
8     b = a*(a+1)/2;
9     cout<<"Sum is : "<<b;
10    return 0;
11 }
```

The 'Output' pane on the right shows the following text:

```
Enter the naural number : 5
Sum is : 15
=== Code Execution Successful ===
```

### 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;
}

```

main.cpp	Output
<pre> 1 #include &lt;iostream&gt; 2 #include &lt;string&gt; 3 4 using namespace std; 5 6 int main() { 7     string inputNumber; 8     cout &lt;&lt; "Enter a number: "; 9     cin &gt;&gt; inputNumber; 10 11     int digitCount = inputNumber.length() - (inputNumber[0] == '-' ? 1 12         : 0); 13     cout &lt;&lt; "The number of digits in the entered number is: " &lt;&lt; 14         digitCount &lt;&lt; endl; 15     return 0; 16 } </pre>	<pre> Enter a number: 12345 The number of digits in the entered number is: 5  === Code Execution Successful === </pre>

### 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;

    return 0;
}

```

main.cpp	Run	Output
<pre> 1 #include &lt;iostream&gt; 2 using namespace std; 3 float area(int r) 4 { 5     float result = 3.14159 * r * r; 6     return result; 7 } 8 int area(int a, int b) 9 { 10     int result = a * b; 11     return result; 12 } 13 float area(float base, float height) 14 { 15     float result = 0.5 * base * height; 16     return result; 17 } 18 int main() 19 { 20     int radius; </pre>		<pre> Enter the radius of the circle: 5 Area of the circle: 78.5397 Enter the length and width of the rectangle: 4 6 Area of the rectangle: 24 Enter the base and height of the triangle: 3 7 Area of the triangle: 10.5  === Code Execution Successful === </pre>

#### 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;
}

```

```

1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 class Account {
5 protected:
6     double balance;
7 public:
8     Account(double bal) : balance(bal) {}
9     virtual void calculateInterest() = 0;
10    virtual ~Account() {}
11 };
12 class SavingsAccount : public Account {
13     double rate;
14     int time;
15 public:
16     SavingsAccount(double bal, double r, int t) : Account(bal), rate
(r), time(t) {}
17     void calculateInterest() override {
18         double interest = balance * (rate / 100) * time;
19         cout << "Savings Account Interest: " << fixed <<
setprecision(2) << interest << endl;
20     }
21 };
22 class CurrentAccount : public Account {
23     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;
}

```

main.cpp	Output
<pre> 1 #include &lt;iostream&gt; 2 #include &lt;iomanip&gt; 3 #include &lt;string&gt; 4 using namespace std; 5 class Employee { 6 protected: 7     string name; 8     int id; 9     double salary; 10 public: 11     Employee(string empName, int empId, double empSalary) 12         : name(empName), id(empId), salary(empSalary) {} 13     virtual void calculateEarnings() = 0; 14     virtual ~Employee() {} 15 }; 16 class Manager : public Employee { 17     int performanceRating; 18 public: 19     Manager(string empName, int empId, double empSalary, int rating) 20         : Employee(empName, empId, empSalary), performanceRating 21         (rating) {} 22     void calculateEarnings() override { 23         double bonus = (performanceRating * 0.1) * salary; 24         double totalEarnings = salary + bonus; 25         cout &lt;&lt; "Manager Details:" &lt;&lt; endl; 26         cout &lt;&lt; "Name: " &lt;&lt; name &lt;&lt; endl; </pre>	<pre> Enter Employee Type (1 for Manager, 2 for Developer): 1 Enter Name: Alice Enter ID: 101 Enter Salary: 50000 Enter Performance Rating (1-5): 4 Manager Details: Name: Alice ID: 101 Base Salary: \$50000.00 Bonus: \$20000.00 Total Earnings: \$70000.00  === Code Execution Successful === </pre>

## 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;
}

```

main.cpp	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  using namespace std; 3  int main() { 4      int i, n; 5      bool is_prime = true; 6      cout &lt;&lt; "Enter a positive integer: "; 7      cin &gt;&gt; n; 8      if (n == 0    n == 1) { 9          is_prime = false; 10     } 11     for (i = 2; i &lt;= n/2; ++i) { 12         if (n % i == 0) { 13             is_prime = false; 14             break; 15         } 16     } 17     if (is_prime) 18         cout &lt;&lt; n &lt;&lt; " is a prime number"; 19     else 20         cout &lt;&lt; n &lt;&lt; " is not a prime number"; 21 22     return 0; 23 } </pre>	Run	<pre> Enter a positive integer: 7 7 is a prime number  === Code Execution Successful === </pre>

## 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;
}

```

main.cpp	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  using namespace std; 3  int main() { 4      int n,i; 5      cout&lt;&lt;"Enter a number : "; 6      cin&gt;&gt;n; 7      for(i=0; i&lt;=n; i++) 8      { 9          if(i%2 != 0) 10         { 11             cout&lt;&lt;" "&lt;&lt;i++; 12         } 13     } 14     return 0; 15 } </pre>	Run	<pre> Enter a number : 10 1 3 5 7 9  === Code Execution Successful === </pre>

## 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;

}

```

main.cpp	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  using namespace std; 3  int main() { 4      int n,i,sum = 0; 5      cout&lt;&lt;"Enter a number : "; 6      cin&gt;&gt;n; 7      for(i=0; i&lt;=n; i++) 8      { 9          if(i%2 != 0) 10         { 11             cout&lt;&lt;" "&lt;&lt;i++; 12         } 13     } 14     cout&lt;&lt;endl; 15     for(i=1;i&lt;=n;i+=2) 16     { 17         sum+=i; 18     } 19     cout&lt;&lt;"Sum is "&lt;&lt;sum; 20     return 0; 21 } </pre>	<div>Run</div>	<pre> Enter a number : 10 1 3 5 7 9  === Code Execution Successful === </pre>

## 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;
}

```

main.cpp	Output
<pre> 1  #include &lt;iostream&gt; 2  using namespace std; 3  int main() 4  { 5      int a,i; 6      cout&lt;&lt;"Enter Number : "; 7      cin&gt;&gt;a; 8      for(i=1;i&lt;=10;i++) 9      { 10         cout&lt;&lt;a&lt;&lt;" * "&lt;&lt;i&lt;&lt;" = "&lt;&lt;i*a&lt;&lt;endl; 11     } 12     return 0; 13 } </pre>	<pre> Enter Number : 3 3 * 1 = 3 3 * 2 = 6 3 * 3 = 9 3 * 4 = 12 3 * 5 = 15 3 * 6 = 18 3 * 7 = 21 3 * 8 = 24 3 * 9 = 27 3 * 10 = 30  === Code Execution Successful === </pre>

## 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;

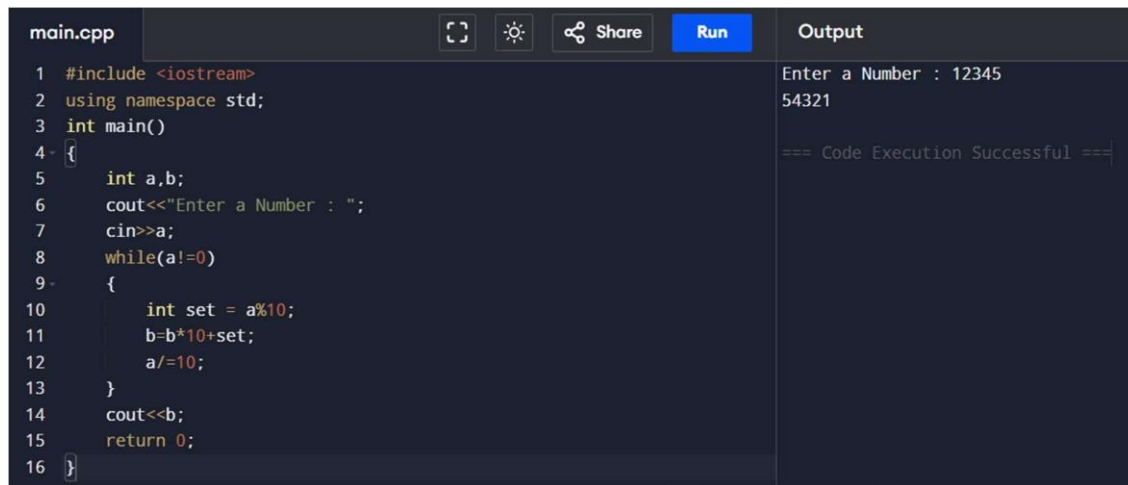
        b=b*10+set;

        a/=10;
    }

    cout<<b;
}

```

```
    return 0;
}
```



```
main.cpp
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a,b;
6      cout<<"Enter a Number : ";
7      cin>>a;
8      while(a!=0)
9      {
10         int set = a%10;
11         b=b*10+set;
12         a/=10;
13     }
14     cout<<b;
15     return 0;
16 }
```

Output

```
Enter a Number : 12345
54321

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

### 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;
```

}

main.cpp	Run	Output
<pre>1 #include &lt;iostream&gt; 2 using namespace std; 3 int main() { 4     int number, largestDigit = 0; 5     cout &lt;&lt; "Enter a positive number: "; 6     cin &gt;&gt; number; 7 8     while (number &gt; 0) { 9         int digit = number % 10; 10        if (digit &gt; largestDigit) 11        { 12            largestDigit = digit; 13        } 14        number /= 10; 15    } 16    cout &lt;&lt; "The largest digit is " &lt;&lt; largestDigit &lt;&lt; endl; 17    return 0; 18 }</pre>		<pre>Enter a positive number: 2734 The largest digit is 7  === Code Execution Successful ===</pre>

## 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	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  using namespace std; 3  int main() 4  { 5      int a,b,c; 6      cout&lt;&lt;"Enter a Number : "; 7      cin&gt;&gt;a; 8      int n=a; 9      while(a!=0) 10     { 11         int set = a%10; 12         b=b*10+set; 13         a/=10; 14     } 15     cout&lt;&lt;b; 16     if(n==b) 17     { 18         cout&lt;&lt;" Is Palindrome "; 19     } 20     else 21         cout&lt;&lt;" Not Palindrome "; 22     return 0; 23 } </pre>	<p>Enter a Number : 121</p> <p>121 Is Palindrome</p> <p>=== Code Execution Successful ===</p>	

### 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;

}

```

main.cpp	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  using namespace std; 3  int main() 4  { 5      int a,i,count; 6      cout&lt;&lt;"Enter a number : "; 7      cin&gt;&gt;a; 8      while(a&gt;0) 9      { 10         count += a%10; 11         a /=10; 12     } 13     cout&lt;&lt;"Sum is "&lt;&lt;count; 14     return 0; 15 } </pre>	<pre> Enter a number : 12345 Sum is 15  === Code Execution Successful === </pre>	

#### 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<<"Employee : "<<Salary(50000,20000)<<endl;

    cout<<"Manager : "<<Salary(80000,30000,20000)<<endl;

    return 0;
}

```

main.cpp	Run	Output
<pre> 1  #include &lt;iostream&gt; 2  #include &lt;string&gt; 3  using namespace std; 4  int Salary(int income) 5  { 6      return income; 7  } 8  int Salary(int base,int bonus) 9  { 10     return base+bonus; 11 } 12 int Salary(int base,int bonus,int performance) 13 { 14     return base+bonus+performance; 15 } 16 int main() 17 { 18     cout&lt;&lt;"Intern Salary : "&lt;&lt;Salary(10000)&lt;&lt;endl; 19     cout&lt;&lt;"Employee : "&lt;&lt;Salary(50000,20000)&lt;&lt;endl; 20     cout&lt;&lt;"Manager : "&lt;&lt;Salary(80000,30000,20000)&lt;&lt;endl; 21     return 0; 22 } </pre>	Run	<pre> Intern Salary : 10000 Employee : 70000 Manager : 130000  === Code Execution Successful === </pre>

## 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  [Icons]  Share  Run  Output
1  #include <iostream>
2  #include <string>
3  using namespace std;
4  class Employee
5  {
6      private:
7          int EID;
8          string EName;
9          float Salary;
10         friend void detail();
11     };
12     void detail()
13     {
14         Employee a1;
15         cout<<"Enter Employee ID : ";
16         cin>>a1.EID;
17         cin.ignore();
18         cout<<"Enter Employee Name : ";
19         getline(cin, a1.EName);
20         cout<<"Enter Salary : ";
21         cin>>a1.Salary;
22         cout<<"Employee ID : "<<a1.EID<<endl;
23         cout<<"Employee Name : "<<a1.EName<<endl;
24         cout<<"Employee Salary : "<<a1.Salary<<endl;
    }
```

```
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:-

```
#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();
}

```

main.cpp

Share

Run

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 class Student
5 {
6     public:
7     int Rnumber;
8     string Name;
9     void input()
10    {
11        cout<<"Enter Roll Number : ";
12        cin>>Rnumber;
13        cin.ignore();
14        cout<<"Enter Name : ";
15        getline(cin, Name);
16    }
17    void display1()
18    {
19        cout<<"Roll Number " <<Rnumber<<endl;
20        cout<<"Name " <<Name<<endl;
21    }
22 };
23 class Result:Student
24 {
25     public:
26     int a,b,c,sum;
```

Output

```
Enter Roll Number : 101
Enter Name : Alice Smith
Marks = 85 90 80
Roll Number 101
Name Alice Smith
Total = 255
Percentage : 85%

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

```

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;
}

```

The screenshot shows a C++ IDE with a file named 'main.cpp'. The code defines a base class 'Shape' with a virtual 'getArea()' method. Two derived classes, 'Rectangle' and 'Circle', override this method. The 'Rectangle' class has private members 'length' and 'breadth', and the 'Circle' class has a private member 'radius'. The main function prompts the user to enter values for a circle, a rectangle, and a triangle, then calculates and displays their areas. The output window shows the program running successfully with the following input and output:

```

Enter Radius for Circle: 7
Enter Length and Breadth for Rectangle: 10 5
Enter Base and Height for Triangle: 8 6
Circle Area: 153.94
Rectangle Area: 50.00
Triangle Area: 24.00

=== Code Execution Successful ===

```

## 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;

}

```

main.cpp	Output
<pre> 1  #include &lt;iostream&gt; 2  #include &lt;cmath&gt; 3  using namespace std; 4  class Shape { 5  public: 6      virtual void input() = 0; 7      virtual void calculateArea() = 0; 8      virtual void displayArea() = 0; 9      virtual ~Shape() {} 10 }; 11 class Circle : public Shape { 12 private: 13     float radius, area; 14 public: 15     void input() override { cin &gt;&gt; radius; } 16     void calculateArea() override { area = M_PI * radius * radius; } 17     void displayArea() override { cout &lt;&lt; "Area of Circle: " &lt;&lt; area 18                                     &lt;&lt; endl; } 19 }; 20 class Rectangle : public Shape { 21 private: 22     float length, breadth, area; 23 public: 24     void input() override { cin &gt;&gt; length &gt;&gt; breadth; } 25     void calculateArea() override { area = length * breadth; } 26     void displayArea() override { cout &lt;&lt; "Area of Rectangle: " &lt;&lt; </pre>	<pre> Enter radius for Circle: 5 Enter length and breadth for Rectangle: 4 6 Enter base and height for Triangle: 3 7 Area of Circle: 78.5398 Area of Rectangle: 24 Area of Triangle: 10.5  === Code Execution Successful === </pre>

## 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;
}

```

main.cpp	Output
<pre> 1 #include &lt;iostream&gt; 2 #include &lt;vector&gt; 3 using namespace std; 4 void operate(vector&lt;vector&lt;int&gt;&gt;&amp; A, vector&lt;vector&lt;int&gt;&gt;&amp; B, vector   &lt;vector&lt;int&gt;&gt;&amp; result, int m, int n) { 5     for (int i = 0; i &lt; m; ++i) { 6         for (int j = 0; j &lt; n; ++j) { 7             result[i][j] = A[i][j] + B[i][j]; 8         } 9     } 10 } 11 void operate(vector&lt;vector&lt;int&gt;&gt;&amp; A, vector&lt;vector&lt;int&gt;&gt;&amp; B, vector   &lt;vector&lt;int&gt;&gt;&amp; result, int m, int n, int p) { 12     for (int i = 0; i &lt; m; ++i) { 13         for (int j = 0; j &lt; p; ++j) { 14             result[i][j] = 0; 15             for (int k = 0; k &lt; n; ++k) { 16                 result[i][j] += A[i][k] * B[k][j]; 17             } 18         } 19     } 20 } 21 void printMatrix(vector&lt;vector&lt;int&gt;&gt;&amp; matrix, int m, int n) { 22     for (int i = 0; i &lt; m; ++i) { 23         for (int j = 0; j &lt; n; ++j) { 24             cout &lt;&lt; matrix[i][j] &lt;&lt; " "; </pre>	<pre> Enter rows and columns for matrix A: 2 2 Enter elements of matrix A: 1 2 3 4 Enter rows and columns for matrix B: 2 2 Enter elements of matrix B: 5 6 7 8 Enter operation type (1 for addition, 2 for multiplication): 1 Matrix Addition Result: 6 8 10 12  === Code Execution Successful === </pre>

## 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;
}
```



```
main.cpp  [Icons]  [Share]  [Run]  [Clear]

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 class Shape {
5 public:
6     virtual double getArea() = 0;
7     virtual ~Shape() {}
8 };
9 class Rectangle : public Shape {
10 private:
11     double length, breadth;
12 public:
13     Rectangle(double l, double b) : length(l), breadth(b) {}
14     double getArea() override {
15         return length * breadth;
16     }
17 };
18 class Circle : public Shape {
19 private:
20     double radius;
21 public:
22     Circle(double r) : radius(r) {}
23     double getArea() override {
24         return M_PI * radius * radius;
25     }
26 };
27
```

```
Output

Enter the shape type (1 for Rectangle, 2 for Circle, 3 for Triangle): 1
Enter Length and Breadth for Rectangle: 5 4
Area of the shape: 20

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

## 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;
}

```

```
main.cpp  Run  Clear
1 #include<iostream>
2 #include<string>
3 using namespace std;
4 class Book
5 {
6     public:
7     string Title, Author;
8     int ISBN;
9     void input()
10    {
11        cout<<"Title: ";
12        getline(cin, Title);
13        cout<<"Author: ";
14        getline(cin, Author);
15        cout<<"ISBN: ";
16        cin>>ISBN;
17    }
18    void display()
19    {
20        cout<<"\n"<<"Title"<<"\n" by "<<Author<<" (ISBN:"<<ISBN<<")";
21    }
22 };
23 class Borrower
24 {
25     public:
26     string Name;
27     int ID;
```

```
Title: C++ Basics
Author: John Doe
ISBN: 1234
Name : Alice
ID: 42
Action: 2
Borrower Alice (ID: 42) has returned "C++ Basics" by John Doe (ISBN:1234).

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

## 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  [Icons]  Share  Run  Output

1  #include <iostream>
2  #include <string>
3  using namespace std;
4  class Vehicle {
5  public:
6      string Brand, Model;
7      double Mileage;
8      void input() {
9          cout << "Brand: ";
10         cin >> Brand;
11         cout << "Model: ";
12         cin >> Model;
13         cout << "Mileage: ";
14         cin >> Mileage;
15     }
16     void output() {
17         cout << "Vehicle: " << Brand << " " << Model << endl;
18         cout << "Mileage: " << Mileage << endl;
19     }
20 };
21 class Car : public Vehicle {
22 public:
23     double Fuel, Distance;
24     void inputCarDetails() {
25         cout << "Fuel (gallons): ";
26         cin >> Fuel;
```

```
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:-

```
#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;

}

```

```

main.cpp
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 class Complex {
6 public:
7     int real, imag;
8     Complex(int r = 0, int i = 0) : real(r), imag(i) {}
9     Complex operator + (const Complex& other) {
10         return Complex(real + other.real, imag + other.imag);
11     }
12     Complex operator * (const Complex& other) {
13         return Complex(real * other.real - imag * other.imag,
14             real * other.imag + imag * other.real);
15     }
16     double magnitude() {
17         return sqrt(real * real + imag * imag);
18     }
19     void display() {
20         if (imag >= 0) {
21             cout << real << " + " << imag << "i" << endl;
22         } else {
23             cout << real << " - " << -imag << "i" << endl;
24         }
25     }

```

Output

```

Enter operation type (1 for Addition, 2 for Multiplication, 3 for Magnitude): 1
Enter complex number 1 (real imaginary): 3 2
Enter complex number 2 (real imaginary): 1 -4
Result: 4 - 2i

=== Code Execution Successful ===

```

## 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;
    }

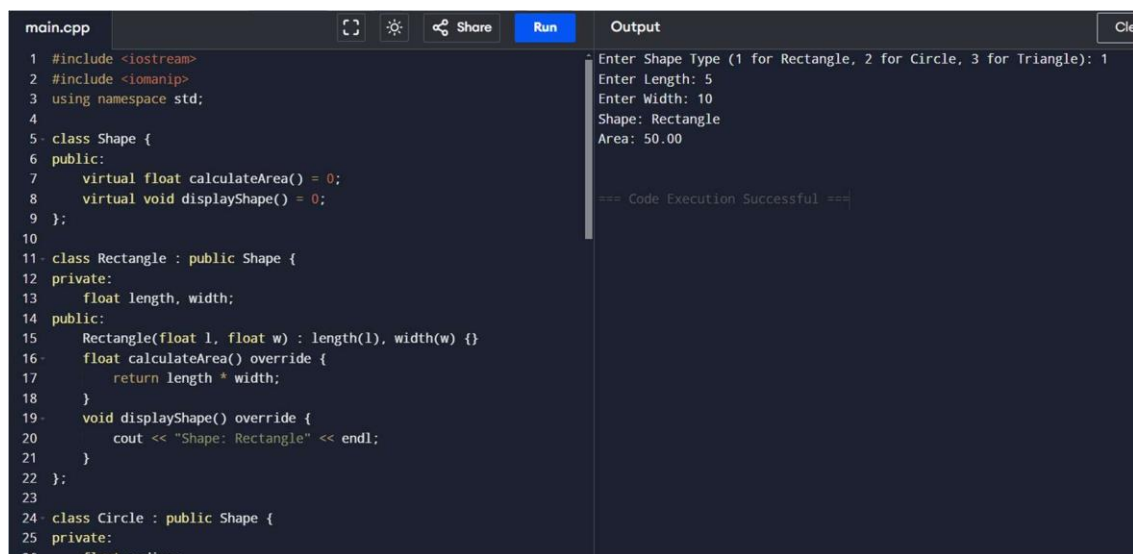
    shape->displayShape();

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

    delete shape;

    return 0;
}

```



The screenshot shows a C++ IDE with a dark theme. The left pane displays the code for `main.cpp`, which defines a base class `Shape` with virtual functions `calculateArea()` and `displayShape()`. It also defines two derived classes: `Rectangle` and `Circle`. The `Rectangle` class has private attributes `length` and `width`, and overrides `calculateArea()` to return `length * width`. The `displayShape()` method in `Rectangle` prints "Shape: Rectangle". The right pane shows the output of the program, which prompts the user to enter a shape type (1 for Rectangle, 2 for Circle, 3 for Triangle). The user enters 1, then the program prompts for length and width. The user enters 5 and 10 respectively. The program then outputs "Shape: Rectangle" and "Area: 50.00". Below the output, it says "=== Code Execution Successful ===".

```

main.cpp
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4
5 class Shape {
6 public:
7     virtual float calculateArea() = 0;
8     virtual void displayShape() = 0;
9 };
10
11 class Rectangle : public Shape {
12 private:
13     float length, width;
14 public:
15     Rectangle(float l, float w) : length(l), width(w) {}
16     float calculateArea() override {
17         return length * width;
18     }
19     void displayShape() override {
20         cout << "Shape: Rectangle" << endl;
21     }
22 };
23
24 class Circle : public Shape {
25 private:
26     float radius;

```

Output

```

Enter Shape Type (1 for Rectangle, 2 for Circle, 3 for Triangle): 1
Enter Length: 5
Enter Width: 10
Shape: Rectangle
Area: 50.00

=== Code Execution Successful ===

```

## 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

Output

Clear

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4
5 float calculateArea(float radius) {
6     return 3.14159 * radius * radius;
7 }
8
9 float calculateArea(float length, float breadth) {
10     return length * breadth;
11 }
12
13 float calculateArea(double base, double height) {
14     return 0.5 * base * height;
15 }
16
17 int main() {
18     int choice;
19     cout << "Enter Shape Type (1 for Circle, 2 for Rectangle, 3 for
    Triangle): ";
20     cin >> choice;
21
22     if (choice == 1) {
23         float radius;
24         cout << "Enter Radius: ";
```

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 ===