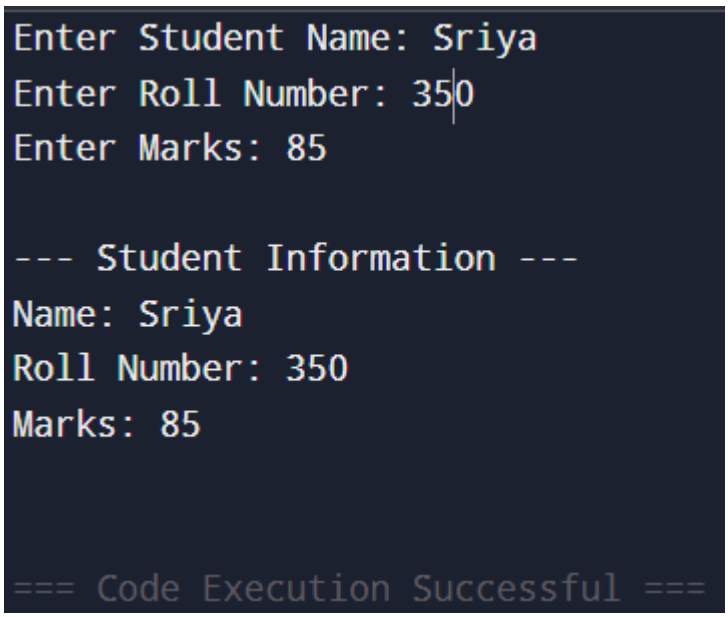
01.DISPLAY STUDENT INFORMATION USING CLASS AND OBJECTS

```
#include <iostream>
using namespace std;
class Student
{
public:
    string name;
    int roll;
    float marks;
    void getData()
{
        cout << "Enter Student Name: ";
        cin >> name;
        cout << "Enter Roll Number: ";
        cin >> roll;
        cout << "Enter Marks: ";
        cin >> marks;
    }
    void display()
  {
        cout << "\n--- Student Information ---\n";
        cout << "Name: " << name << endl;
        cout << "Roll Number: " << roll << endl;
        cout << "Marks: " << marks << endl;
    }
```

```
};
int main()
{
    Student s;
    s.getData();
    s.display();
    return 0;
}
```

OUTPUT:

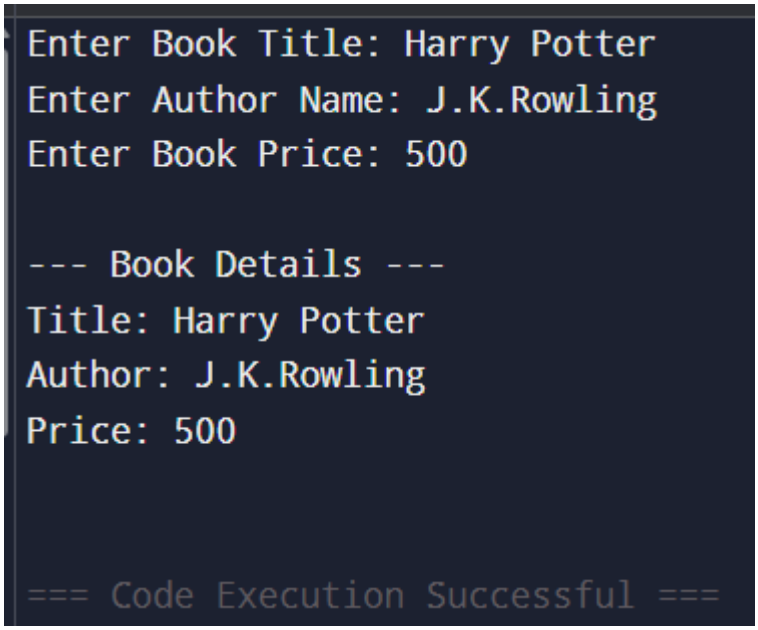


02. BOOK DETAILS (function defined outside class)

```
#include <iostream>
using namespace std;
class Book
{
public:
    string title;
    string author;
    float price;
    void getData();
    void display();
};
void Book::getData()
{
    cout << "Enter Book Title: ";
    cin >> title;
    cout << "Enter Author Name: ";
    cin >> author;
    cout << "Enter Book Price: ";
    cin >> price;
}
void Book::display()
{
    cout << "\n--- Book Details ---\n";
    cout << "Title: " << title << endl;
    cout << "Author: " << author << endl;
```

```
    cout << "Price: " << price << endl;
}
int main()
{
    Book b;
    b.getData();
    b.display();
   return 0;
}
```

OUTPUT:

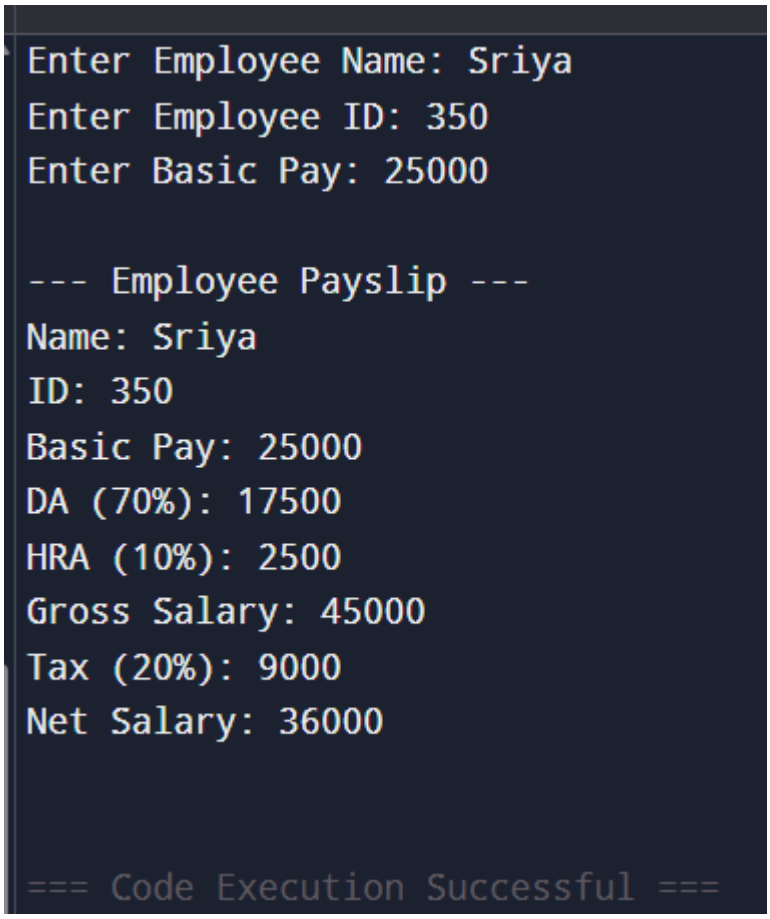


03.EMPLOYEE PAYSLIP USING CLASS&OBJECTS

```
#include <iostream>
using namespace std;
class Employee
{
public:
    string name;
    int id;
    float basicPay;
    float da, hra, gross, tax, net;
   void getData();
    void calculate();
    void display();
};
void Employee::getData()
{
    cout << "Enter Employee Name: ";
    cin >> name;
    cout << "Enter Employee ID: ";
    cin >> id;
    cout << "Enter Basic Pay: ";
    cin >> basicPay;
}
void Employee::calculate()
{
    da = basicPay * 0.70;
```

```
    hra = basicPay * 0.10;
    gross = basicPay + da + hra;
    tax = gross * 0.20;
    net = gross - tax;
}
void Employee::display()
{
    cout << "\n--- Employee Payslip ---\n";
    cout << "Name: " << name << endl;
    cout << "ID: " << id << endl;
    cout << "Basic Pay: " << basicPay << endl;
    cout << "DA (70%): " << da << endl;
    cout << "HRA (10%): " << hra << endl;
    cout << "Gross Salary: " << gross << endl;
    cout << "Tax (20%): " << tax << endl;
    cout << "Net Salary: " << net << endl;
}
int main() {
    Employee e;
   e.getData();
    e.calculate();
    e.display();
    return 0;
}
```

OUTPUT:

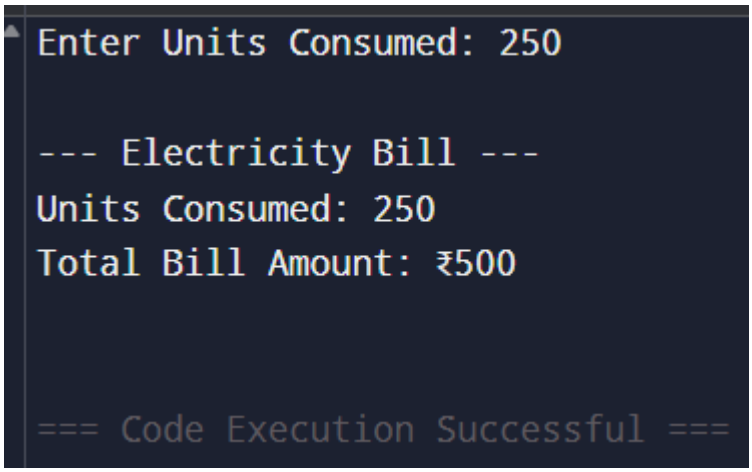

Enter Employee Name: Sriya
Enter Employee ID: 350
Enter Basic Pay: 25000
--- Employee Payslip ---
Name: Sriya
ID: 350
Basic Pay: 25000
DA (70%): 17500
HRA (10%): 2500
Gross Salary: 45000
Tax (20%): 9000
Net Salary: 36000
=== Code Execution Successful ===

04.ELECTRICITY BILL

```
#include <iostream>
using namespace std;
class Electricity
{
public:
   int units;
   float bill;
   void getData();
   void calculate();
   void display();
};
void Electricity::getData()
{
   cout << "Enter Units Consumed: ";
   cin >> units;
}
void Electricity::calculate()
{
   if (units <= 100)
{
      bill = units * 1.50;
   }
   else if (units <= 200)
{
      bill = (100 * 1.50) + (units - 100) * 2.00;
```

```
    }
    else {
        bill = (100 * 1.50) + (100 * 2.00) + (units - 200) * 3.00;
    }
}
void Electricity::display()
{
    cout << "\n--- Electricity Bill ---\n";
    cout << "Units Consumed: " << units << endl;
    cout << "Total Bill Amount: ₹" << bill << endl;
}
int main()
{
    Electricity e;
    e.getData();
    e.calculate();
    e.display();
    return 0;
}
```

OUTPUT:


Enter Units Consumed: 250
--- Electricity Bill ---
Units Consumed: 250
Total Bill Amount: ₹500
=== Code Execution Successful ===

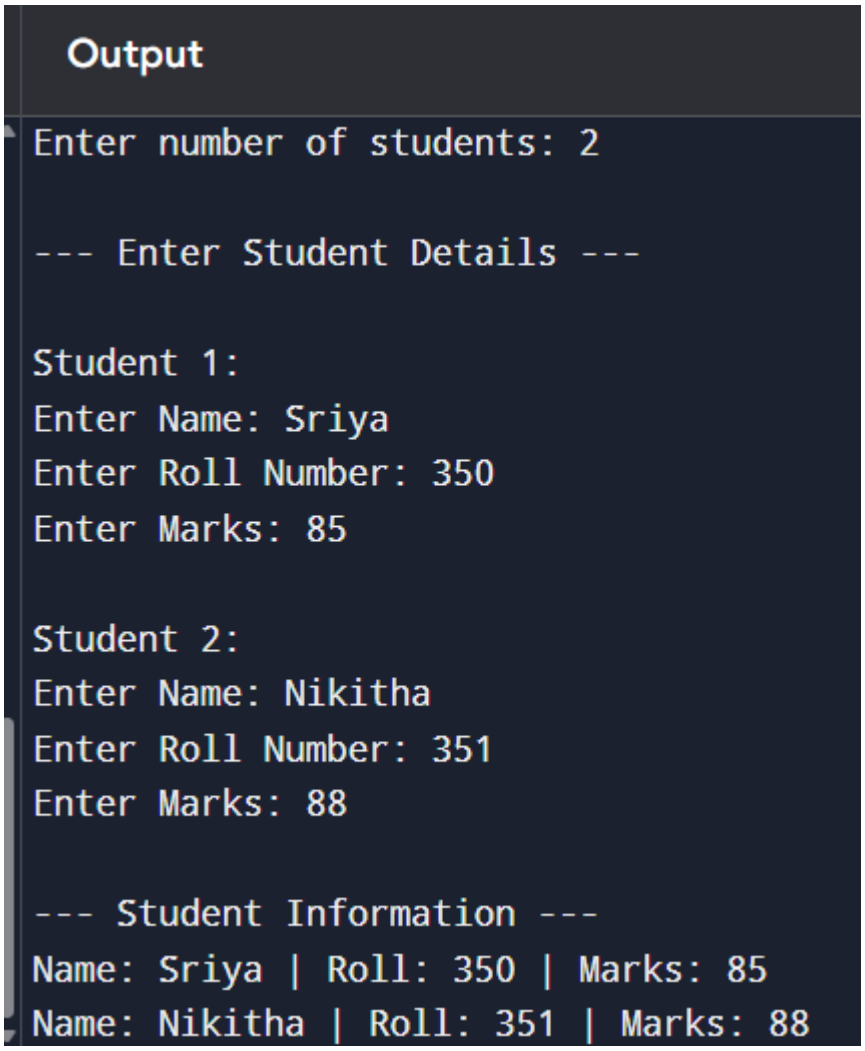
05.STUDENT INFORMATION FOR N STUDENTS

(Array of objects)

```
#include <iostream>
using namespace std;
class Student
{
public:
    string name;
    int roll;
    float marks;
   void getData();
    void display();
};
void Student::getData()
{
    cout << "Enter Name: ";
    cin >> name;
    cout << "Enter Roll Number: ";
    cin >> roll;
    cout << "Enter Marks: ";
    cin >> marks;
}
void Student::display()
{
    cout << "Name: " << name
         << " | Roll: " << roll
```

```
        << " | Marks: " << marks << endl;
}
int main()
{
    int n;
    cout << "Enter number of students: ";
    cin >> n;
   Student s[n];
  cout << "\n--- Enter Student Details ---\n";
    for (int i = 0; i < n; i++) {
        cout << "\nStudent " << i + 1 << ":\n";
        s[i].getData();
    }
  cout << "\n--- Student Information ---\n";
    for (int i = 0; i < n; i++) {
        s[i].display();
    }
  return 0;
}
```

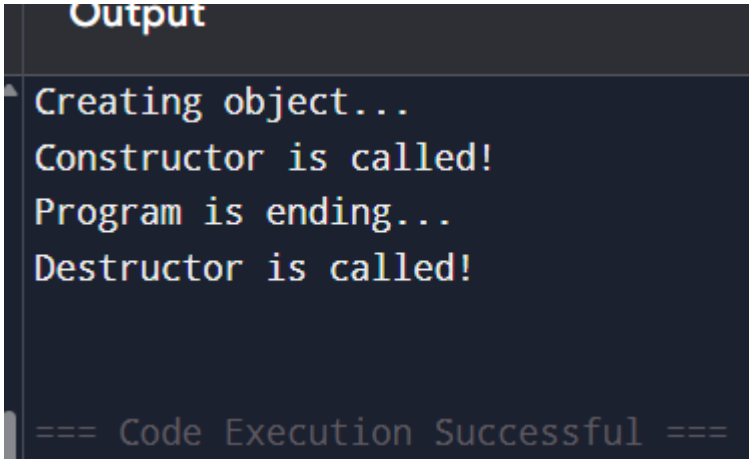
OUTPUT:


Output
Enter number of students: 2
--- Enter Student Details ---
Student 1:
Enter Name: Sriya
Enter Roll Number: 350
Enter Marks: 85
Student 2:
Enter Name: Nikitha
Enter Roll Number: 351
Enter Marks: 88
--- Student Information ---
Name: Sriya | Roll: 350 | Marks: 85
Name: Nikitha | Roll: 351 | Marks: 88

06. CONSTRUCTOR AND DESTRUCTOR

```
#include <iostream>
using namespace std;
class Demo
{
public:
   Demo()
{
       cout << "Constructor is called!" << endl;
   }
   ~Demo()
{
       cout << "Destructor is called!" << endl;
   }
};
int main()
{
   cout << "Creating object..." << endl;
   Demo obj;
  cout << "Program is ending..." << endl;
   return 0;
}
```

OUTPUT:


Output
Creating object...
Constructor is called!
Program is ending...
Destructor is called!
=== Code Execution Successful ===

07. CONSTRUCTOR OVERLOADING

```
#include <iostream>
using namespace std;
class Demo
{
public:
   int a, b;
   Demo()
{
      a = 0;
      b = 0;
      cout << "Default Constructor Called" << endl;
   }
   Demo(int x)
{
      a = x;
      b = 0;
      cout << "Constructor with 1 parameter Called" << endl;
   }
   Demo(int x, int y)
{
      a = x;
      b = y;
      cout << "Constructor with 2 parameters Called" << endl;
   }
```

```
    void display()
{
        cout << "a = " << a << ", b = " << b << endl;
    }
};
int main()
{
    Demo d1;
    d1.display();
   Demo d2(10);
    d2.display();
   Demo d3(20, 30);
    d3.display();
    return 0;
}
```

OUTPUT:

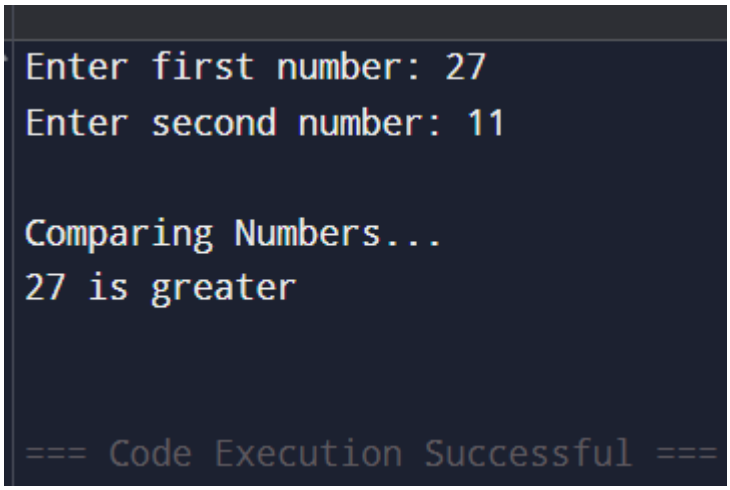
```
Default Constructor Called
a = 0, b = 0
Constructor with 1 parameter Called
a = 10, b = 0
Constructor with 2 parameters Called
a = 20, b = 30


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

08. OBJECT AS FUNCTION ARGUMENT

```
#include <iostream>
using namespace std;
class Number
{
public:
   int value;
 void getData(string msg)
{
       cout << msg;
       cin >> value;
   }
};
void compare(Number n1, Number n2)
{
   cout << "\nComparing Numbers...\n";
   if (n1.value > n2.value)
       cout << n1.value << " is greater\n";
   else if (n2.value > n1.value)
       cout << n2.value << " is greater\n";
   else
       cout << "Both numbers are equal\n";
}
int main() {
   Number a, b;
```

```
    a.getData("Enter first number: ");
    b.getData("Enter second number: ");
   compare(a, b);
   return 0;
}
```

OUTPUT:

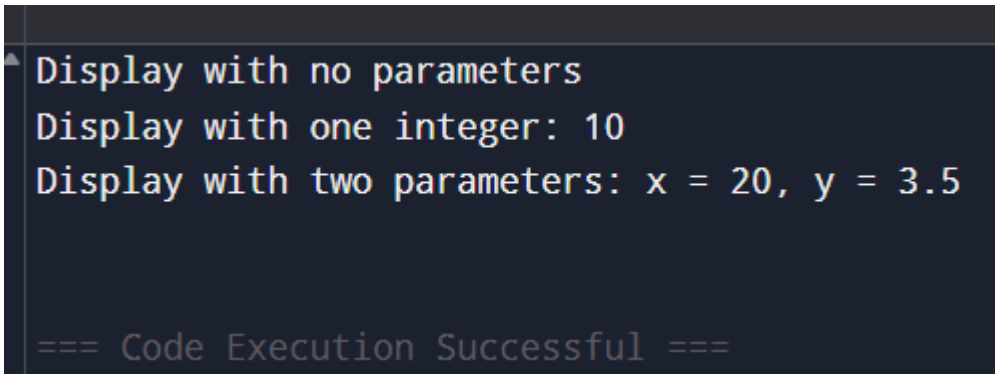


09.FUNCTION OVERLOADING

```
#include <iostream>
using namespace std;
class Overload
{
public:
   void display()
{
       cout << "Display with no parameters" << endl;
   }
   void display(int x)
{
       cout << "Display with one integer: " << x << endl;
   }
   void display(int x, float y)
{
       cout << "Display with two parameters: ";
       cout << "x = " << x << ", y = " << y << endl;
   }
};
int main()
{
   Overload obj;
  obj.display();
   obj.display(10);
   obj.display(20, 3.5f);
```

```
    return 0;
}
```

OUTPUT:

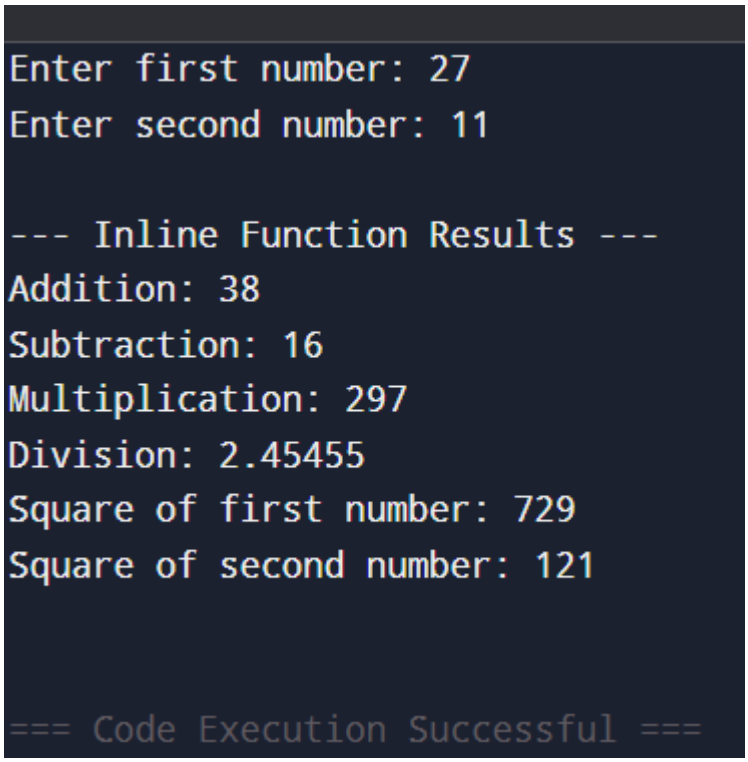


10. INLINE FUNCTION

```
#include <iostream>
using namespace std;
inline int add(int a, int b)
{
   return a + b;
}
inline int subtract(int a, int b)
{
   return a - b;
}
inline int multiply(int a, int b)
{
   return a * b;
}
inline float divide(float a, float b)
{
   return a / b;
}
inline int square(int x)
{
   return x * x;
}
int main()
{
   int x, y;
```

```
    cout << "Enter first number: ";
    cin >> x;
   cout << "Enter second number: ";
    cin >> y;
   cout << "\n--- Inline Function Results ---\n";
   cout << "Addition: " << add(x, y) << endl;
    cout << "Subtraction: " << subtract(x, y) << endl;
    cout << "Multiplication: " << multiply(x, y) << endl;
    if (y != 0)
        cout << "Division: " << divide(x, y) << endl;
    else
        cout << "Division: Not possible (div by 0)" << endl;
     cout << "Square of first number: " << square(x) << endl;
    cout << "Square of second number: " << square(y) << endl;
  return 0;
}
```

OUTPUT:


Enter first number: 27
Enter second number: 11
--- Inline Function Results ---
Addition: 38
Subtraction: 16
Multiplication: 297
Division: 2.45455
Square of first number: 729
Square of second number: 121
=== Code Execution Successful ===

11. MANIPULATORS

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a = 25, b = 1234;
    float x = 45.6789;
   cout << "--- Demonstrating Manipulators ---" << endl;
    cout << "Using setw(): " << endl;
    cout << setw(5) << a << endl;
    cout << setw(5) << b << endl;
    cout << "\nUsing setprecision() and fixed:" << endl;
    cout << fixed << setprecision(2);
    cout << "Value of x = " << x << endl;
    cout << "\nUsing showpoint:" << endl;
    cout << showpoint << 45.0 << endl;
    cout << "\nUsing endl:" << endl;
    cout << "This is line 1" << endl;
    cout << "This is line 2" << endl;
   return 0;
}
```

OUTPUT:

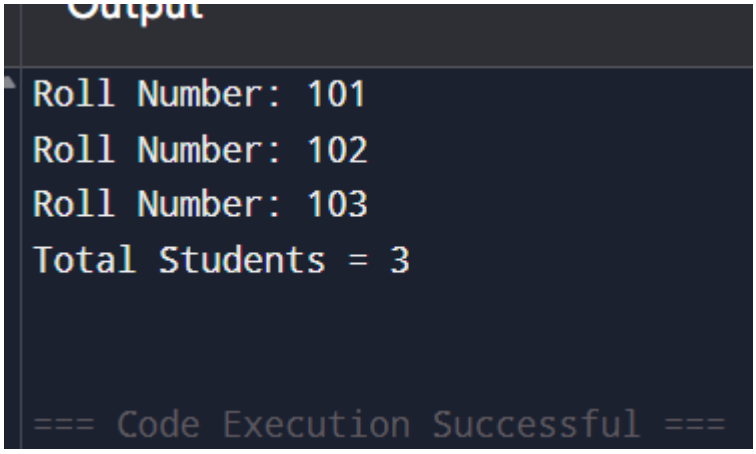
```
--- Demonstrating Manipulators ---
Using setw():
   25
 1234

Using setprecision() and fixed:
Value of x = 45.68

Using showpoint:
45.00

Using endl:
This is line 1
This is line 2


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

12. STATIC DATA MEMBER AND STATIC DATA FUNCTION

```
#include <iostream>
using namespace std;
class Student
{
private:
    int roll;
    static int count;
public:
    void getData(int r)
{
        roll = r;
        count++;
    }
   void display()
{
        cout << "Roll Number: " << roll << endl;
    }
  static void showCount()
{
        cout << "Total Students = " << count << endl;
    }
};
int Student::count = 0;
int main()
{
```

```
    Student s1, s2, s3;
    s1.getData(101);
    s2.getData(102);
    s3.getData(103);
    s1.display();
    s2.display();
    s3.display();
    Student::showCount();
     return 0;
}
```

OUTPUT:

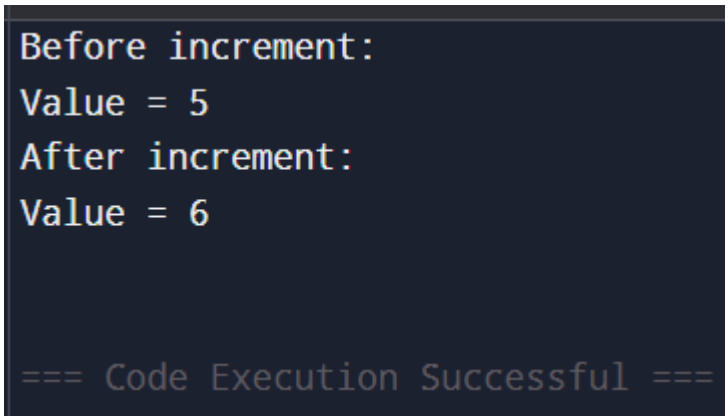


13.UNARY OPERATOR OVERLOADING

```
#include <iostream>
using namespace std;
class Number
{
private:
   int value;
public:
   Number(int v)
{
       value = v;
   }
   void operator++()
{
       value = value + 1;
   }
  void display()
{
       cout << "Value = " << value << endl;
   }
};
int main()
{
   Number n1(5);
  cout << "Before increment:" << endl;
   n1.display();
```

```
    ++n1;
    cout << "After increment:" << endl;
     n1.display();
    return 0;
}
```

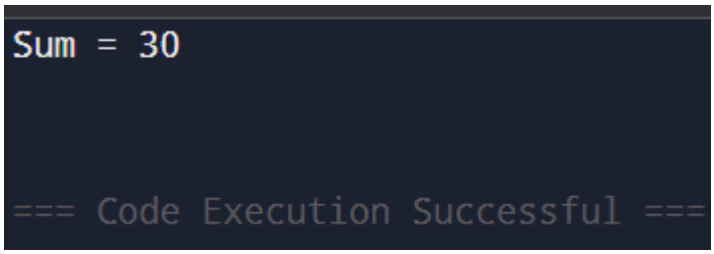
OUTPUT:



14.FRIEND FUNCTION

```
#include <iostream>
using namespace std;
class Sample
{
private:
   int a, b;
public:
   Sample(int x, int y)
{
       a = x;
       b = y;
   }
   friend int add(Sample s);
};
int add(Sample s)
{
   return s.a + s.b;
}
int main()
{
   Sample obj(10, 20);
cout << "Sum = " << add(obj) << endl;

   return 0;
}
```

OUTPUT:

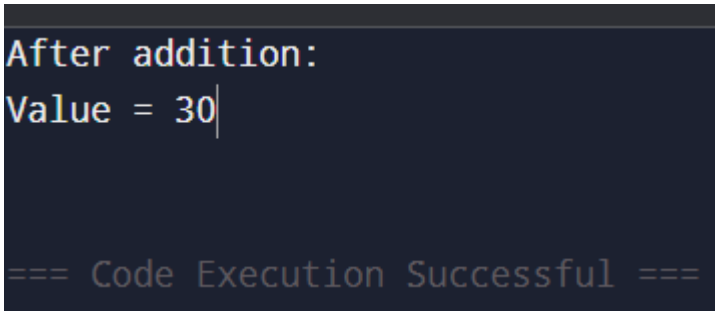


15.BINARY OPERATOR OVERLOADING

```
#include <iostream>
using namespace std;
class Number
{
private:
   int value;
public:
   Number(int v = 0)
{
       value = v;
   }
   Number operator+(Number obj)
{
       Number temp;
       temp.value = value + obj.value;
       return temp;
   }
  void display()
{
       cout << "Value = " << value << endl;
   }
};
int main()
{
   Number n1(10);
```

```
    Number n2(20);
   Number n3 = n1 + n2;
   cout << "After addition:" << endl;
    n3.display();
  return 0;
}
```

OUTPUT:

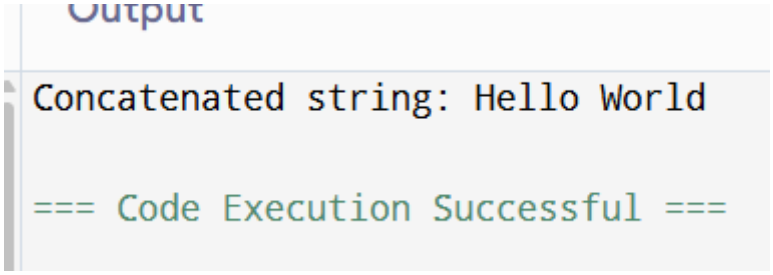


16.STRING CONCATENATION

```
#include <iostream>
#include <cstring>
using namespace std;
class MyString
{
   char str[50];
public:
   MyString() { str[0] = '\0'; }
    MyString(const char s[]) {
      strcpy(str, s);
   }
   MyString operator+(MyString s) {
      MyString temp;
      strcpy(temp.str, str);
      strcat(temp.str, s.str);
      return temp;
   }
    void display() {
      cout << str << endl;
   }
};
int main() {
   MyString s1("Hello ");
   MyString s2("World!");
   MyString s3 = s1 + s2;
```

```
    cout << "Concatenated String: ";
    s3.display();
   return 0;
}
```

OUTPUT:



17.SINGLE INHERITANCE

```
#include <iostream>
using namespace std;
class Animal {
public:
   void eat()
{
       cout << "Animal is eating\n";
   }
};
class Dog : public Animal {
public:
   void bark() {
       cout << "Dog is barking\n";
   }
};
int main() {
   Dog d;
   d.eat();
   d.bark();
   return 0;
}
```

OUTPUT:

```
Animal is eating
Dog is barking


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

18.MULTILEVEL INHERITANCE

```
#include <iostream>
using namespace std;
class GrandParent {
public:
    void showGrandParent() {
        cout << "GrandParent class\n";
    }
};
class Parent : public GrandParent {
public:
    void showParent() {
        cout << "Parent class\n";
    }
};
class Child : public Parent {
public:
    void showChild() {
        cout << "Child class\n";
    }
};
int main() {
    Child c;
    c.showGrandParent();
    c.showParent();
    c.showChild();
```

```
    return 0;
}
```

OUTPUT:

```
GrandParent class
Parent class
Child class


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

19.MULTIPLE INHERITANCE

```
#include <iostream>
using namespace std;
class Teacher {
public:
    void teach() {
        cout << "Teaching students\n";
    }
};
class Researcher {
public:
    void research() {
        cout << "Doing research\n";
    }
};
class Professor : public Teacher, public Researcher {
public:
    void guide() {
        cout << "Guiding students\n";
    }
};
int main() {
    Professor p;
    p.teach();
    p.research();
    p.guide();
```

```
    return 0;
}
```

OUTPUT:

```
Teaching students
Doing research
Guiding students


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

20.MEMORY MANAGEMENT OPERATOR

```
#include <iostream>
#include <cstdlib>
using namespace std;
class Sample {
    int x;
public:
    Sample(int v = 0) : x(v) {}
   void* operator new(size_t size) {
        cout << "Overloaded new, size = " << size << endl;
        void* p = malloc(size);
        if (!p) {
            throw bad_alloc();
        }
        return p;
    }
   void operator delete(void* p) {
        cout << "Overloaded delete\n";
        free(p);
    }
  void show() {
        cout << "x = " << x << endl;
    }
};
int main() {
    Sample* s = new Sample(10);
```

```
    s->show();
    delete s;
    return 0;
}
```

OUTPUT:

```
Overloaded new, size = 4
x = 10
Overloaded delete


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

21.VIRTUAL FUNCTION

```
#include <iostream>
using namespace std;
class Shape {
public:
    virtual void area() {
        cout << "Area of generic shape\n";
    }
};
class Rectangle : public Shape {
    int w, h;
public:
    Rectangle(int width, int height) : w(width), h(height) {}
    void area() override {
        cout << "Area of rectangle = " << w * h << endl;
    }
};
class Circle : public Shape {
    int r;
public:
    Circle(int radius) : r(radius) {}
    void area() override {
        cout << "Area of circle = " << 3.14 * r * r << endl;
    }
};
```

```
int main() {
    Shape *ptr;
    Rectangle rect(4, 5);
    Circle cir(3);
    ptr = &rect;
    ptr->area();
    ptr = &cir;
    ptr->area();
   return 0;
}
```

OUTPUT:

```
rea of rectangle = 20
rea of circle = 28.26


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

22.USING this pointer

```
#include <iostream>
using namespace std;
class Counter {
    int value;
public:
    Counter() : value(0) {}
    Counter& increment(int x) {
        this->value += x;
        return *this;
    }
    Counter& decrement(int x) {
        this->value -= x;
        return *this;
    }
    void show() {
        cout << "Current value = " << value << endl;
    }
};
int main() {
    Counter c;
    c.increment(10).decrement(3).increment(5);
    c.show();
     return 0;
}
```

OUTPUT:

```
Current value = 12


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

23.FUNCTION TEMPLATE

```
#include <iostream>
using namespace std;
template <class T>
T myMax(T a, T b) {
    return (a > b) ? a : b;
}
template <typename T1, typename T2>
class MyPair {
    T1 first;
    T2 second;
public:
    MyPair(T1 f, T2 s) : first(f), second(s) {}
   void show() {
        cout << "First  = " << first << endl;
        cout << "Second = " << second << endl;
    }
};
int main() {
    cout << "Max(3, 7) = " << myMax(3, 7) << endl;
    cout << "Max(4.5, 2.1) = " << myMax(4.5, 2.1) << endl;
    cout << "Max('a', 'z') = " << myMax('a', 'z') << endl;
    MyPair<int, string> p(101, "Rahul");
    p.show();
```

```
    return 0;
}
```

OUTPUT:

```
Max(3, 7) = 7
Max(4.5, 2.1) = 4.5
Max('a', 'z') = z
First  = 101
Second = Rahul


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

24.EXCEPTION HANDLING

```
#include <iostream>
#include <stdexcept>
using namespace std;
double divide(int a, int b) {
    if (b == 0) {
        throw runtime_error("Division by zero is not allowed");
    }
    return (double)a / b;
}
int main() {
    int x, y;
    cout << "Enter two integers (x y): ";
    cin >> x >> y;
    try {
        double result = divide(x, y);
        cout << "Result = " << result << endl;
    }
    catch (const runtime_error &e) {
        cout << "Error: " << e.what() << endl;
    }
   cout << "Program continues after exception handling..." << endl;
    return 0;
}
```

OUTPUT:

```
Enter two integers (x y): 2 3
Result = 0.666667
Program continues after exception handling...


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