NAME :Yash KUMAR

UID :22BCS15424

QUES:-

1. **Sum of two numbers**

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

int main() {
   float num1, num2, sum;

   cout << "Enter first number: ";
cin >> num1;
   cout << "Enter second number: ";
   cin >> num2;

   sum = num1 + num2;

   cout << "The sum of " << num1 << " and " << num2 << " is: " << sum
<< endl;

   return 0;
}
```

```
Enter first number: 23
Enter second number: 55
The sum of 23 and 55 is: 78
```

2. **Sum of all natural numbers**

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

```
int main() {
   int n;

   cout << "Enter a positive integer: ";
cin >> n;

   int sum = n * (n + 1) / 2;

   cout << "The sum of the first " << n << " natural numbers is: " << sum
<< endl;

   return 0;
}
```

```
Enter a positive integer: 29
The sum of the first 29 natural numbers is: 435
```

3. **Find the positive or negative numbers**

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

int main() {
   float num;

   cout << "Enter a number: ";
cin >> num;

   if (num > 0) {
      cout << "The number is positive." << endl;
   } else if (num < 0) {
      cout << "The number is negative." << endl;
   } else {
      cout << "The number is zero." << endl;
   }
```

```
    return 0;
}
```

```
Enter a number: -55
The number is negative.
```

4. **write a c++ program to overload a function max that return the max of two integer and 3 floating value.**

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

int max(int a, int b) {
return (a > b) ? a : b;
}
float max(float a, float b, float c) {
    return (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);
}

int main() {     int
num1, num2;
    cout << "Enter two integers: ";
cin >> num1 >> num2;
    cout << "The maximum of " << num1 << " and " << num2 << " is: " <<
max(num1, num2) << endl;

    float f1, f2, f3;
    cout << "Enter three floating-point numbers: ";
    cin >> f1 >> f2 >> f3;     cout << "The maximum of " << f1 << ", " << f2
<< ", and " << f3 << " is: " << max(f1, f2, f3) << endl;

    return 0;
}
```

```
Enter two integers: 2
6
The maximum of 2 and 6 is: 6
Enter three floating-point numbers: 6.5
4.5
8.6
The maximum of 6.5, 4.5, and 8.6 is: 8.6
```

5. **Write a program to calculate the area of different shapes using function overloading. Implement overloaded functions to compute the area of a circle, a rectangle, and a triangle.**

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

class ShapeArea { public:
   double calculateArea(double radius) {
      return 3.14159 * radius * radius;
   }

   double calculateRectangleArea(double length, double breadth) {
return length * breadth;
   }

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

int main() {
   ShapeArea shape;

   double radius, length, breadth, base, height;
cin >> radius;    cin >> length >> breadth;
   cin >> base >> height;

   // Calculate and output areas
cout << fixed << setprecision(5);
```

```
   cout << shape.calculateArea(radius) << endl;          // Circle    cout
<<   shape.calculateRectangleArea(length,   breadth)   <<   endl;   //
Rectangle
   cout << shape.calculateTriangleArea(base, height) << endl; //
Triangle

   return 0;
}
```

```
4
5 6
5 3
50.26544
30.00000
7.50000
```

6. Write a program to demonstrate runtime polymorphism in C++ using a base class Shape and derived classes Circle, Rectangle, and Triangle. The program should use virtual functions to calculate and print the area of each shape based on user input.

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

class Shape { public:
   virtual double calculateArea() const = 0; // Pure virtual function
virtual ~Shape() {}
};

class Circle : public Shape {
double radius;
public:
   Circle(double r) : radius(r) {}     double
calculateArea() const override {
```

```
        return 3.14159 * radius * radius;
    }
};

class Rectangle : public Shape {
double length, breadth; public:
    Rectangle(double l, double b) : length(l), breadth(b) {}
double calculateArea() const override {
        return length * breadth;
    }
};

class Triangle : public Shape {
double base, height; public:
    Triangle(double b, double h) : base(b), height(h) {}
double calculateArea() const override {
        return 0.5 * base * height;
    }
};

void printArea(const Shape& shape, const string& shapeName) {
cout << "Area of " << shapeName << ": " << fixed << setprecision(5)
<< shape.calculateArea() << endl;
}
int main() {
    double radius, length, breadth, base, height;
cin >> radius;     cin >> length >> breadth;
    cin >> base >> height;

    Circle circle(radius);
    Rectangle rectangle(length, breadth);
    Triangle triangle(base, height);

    // Print areas     printArea(circle,
"Circle");     printArea(rectangle,
"Rectangle");
    printArea(triangle, "Triangle");
```

```
   return 0;
}
```

```
1 2
3 4
5 6
Area of Circle: 3.14159
Area of Rectangle: 6.00000
Area of Triangle: 10.00000
```

7. Create a C++ program to simulate an employee management system using hierarchical inheritance. Design a base class Employee that stores basic details (name, ID, and salary). Create two derived classes:
Manager: Add and calculate bonuses based on performance ratings.
Developer: Add and calculate overtime compensation based on extra hours worked.
The program should allow input for both types of employees and display their total earnings.

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

class Employee {
protected:
string name;
int id;    double
salary; public:
   Employee(const string& name, int id, double salary)
      : name(name), id(id), salary(salary) {}

   virtual void displayDetails() const = 0;
virtual double calculateEarnings() const = 0;
virtual ~Employee() {}
};
```

```
class Manager : public Employee {
   int rating;

public:
   Manager(const string& name, int id, double salary, int rating)
      : Employee(name, id, salary), rating(rating) {}

   double calculateEarnings() const override {
double bonus = (rating * 0.1) * salary;        return
salary + bonus;
   }

   void displayDetails() const override {
double bonus = (rating * 0.1) * salary;
      cout << "Employee: " << name << " (ID: " << id << ")\n";
cout << "Role: Manager\n";        cout << "Base Salary: " <<
salary << "\n";        cout << "Bonus: " << bonus << "\n";
      cout << "Total Earnings: " << calculateEarnings() << "\n";
   }
};

class Developer : public Employee {
   int extraHours;

public:
   Developer(const string& name, int id, double salary, int extraHours)
: Employee(name, id, salary), extraHours(extraHours) {}

   double calculateEarnings() const override {
double overtimeCompensation = extraHours * 500;
return salary + overtimeCompensation;
   }

   void displayDetails() const override {
      double overtimeCompensation = extraHours * 500;
cout << "Employee: " << name << " (ID: " << id << ")\n";
```

```
cout << "Role: Developer\n";        cout << "Base Salary: " <<
salary << "\n";
      cout << "Overtime Compensation: " << overtimeCompensation <<
"\n";
      cout << "Total Earnings: " << calculateEarnings() << "\n";
   }
};

int main() {     int
employeeType;
   cout << "Enter Employee Type (1 for Manager, 2 for Developer): ";
cin >> employeeType;

   if (employeeType == 1) {
      string name;
int id;        double
salary;
      int rating;

      cout << "Enter Name: ";
cin >> name;        cout <<
"Enter ID: ";
      cin >> id;
      cout << "Enter Salary: ";
      cin >> salary;
      cout << "Enter Rating (1-5): ";
cin >> rating;

      if (rating < 1 || rating > 5) {
cout << "Invalid rating." << endl;
         return 0;
      }

      Manager manager(name, id, salary, rating);
manager.displayDetails();     } else if
(employeeType == 2) {
```

```
        string name;
int id;        double
salary;
        int extraHours;

        cout << "Enter Name: ";
cin >> name;        cout <<
"Enter ID: ";        cin >> id;
cout << "Enter Salary: ";
cin >> salary;
        cout << "Enter Extra Hours Worked: ";
        cin >> extraHours;

        if (extraHours < 0 || extraHours > 100) {
            cout << "Invalid extra hours." << endl;
            return 0;
        }

        Developer developer(name, id, salary, extraHours);
        developer.displayDetails();
    } else {
        cout << "Invalid employee type." << endl;
    }

    return 0;
}
```

```
Enter Employee Type (1 for Manager, 2 for Developer): 1
Enter Name: Rohan
Enter ID: 15402
Enter Salary: 500000
Enter Rating (1-5): 5
Employee: Rohan (ID: 15402)
Role: Manager
Base Salary: 500000
Bonus: 250000
Total Earnings: 750000
```