# 1. Problem Title: Object-Oriented Magic

**Problem Statement:**

You are required to create a class `MagicNumber` to manipulate and perform magical operations on numbers. The class should have the following functionalities:

1. **Initialization**: The class should be initialized with an integer `x`.
2. **Method 1: Increment**: Implement a method `void increment()`, which increments the value of `x` by 1.
3. **Method 2: Decrement**: Implement a method `void decrement()`, which decrements the value of `x` by 1.
4. **Method 3: Add**: Implement a method `void add(int y)`, which adds the value of `y` to `x`.
5. **Method 4: Subtract**: Implement a method `void subtract(int y)`, which subtracts the value of `y` from `x`.
6. **Method 5: GetResult**: Implement a method `int getResult()`, which returns the current value of `x`.

Write a C++ class `MagicNumber` with these methods, and ensure that the value of `x` remains within the constraints of -10^9 <= x <= 10^9 after each operation.

**Input:**

- The first line contains an integer `x` (1 <= x <= 10^9).
- The second line contains an integer `q` (1 <= q <= 100), the number of queries.
- The following `q` lines contain one of the following operations:
  - `1 y` (Increment `x` by 1)
  - `2 y` (Decrement `x` by 1)
  - `3 y` (Add `y` to `x`)

- `4 y` (Subtract `y` from `x`)
- `5` (Get the current value of `x`)

**Output:**

For each operation of type `5`, output the result of `x` after the operation.

**Constraints:**

- The value of `x` should be within the range -10^9 <= x <= 10^9 at all times.
- For each operation of type `5`, the result will fit within a 32-bit signed integer.

Here are three test cases to check the solution:

**Test Case 1:**

**Input:**

```
5
6
1 1
2 2
3 10
4 3
5
5
```

**Output:**

```
4
7
```

**Test Case 2:**

**Input:**

```
1000000000
5
3 1000000000
4 500000000
1 5
```

```
2 10
5
```

### Output:

```
1500000000
1499999990
```

### Test Case 3:

### Input:

```
-1000000000
3
4 500000000
2 10
5
```

### Output:

```
-999999990
```

# 2. Problem Title: Protected Treasur

**Problem Statement:**

You are given a treasure map with a set of locations containing hidden treasures. Your task is to design a C++ class `TreasureMap` to encapsulate the treasure data. The class should have the following functionalities:

1. **Initialization**: The class should be initialized with an integer `n`, the number of treasure locations.
2. **Method 1: AddTreasure**: Implement a method `void AddTreasure(int location, int value)`, which adds a treasure with a given `location` and `value`. You should ensure that no two treasures can be added at the same location.
3. **Method 2: GetTreasureValue**: Implement a method `int GetTreasureValue(int location)`, which returns the value of the treasure located at a specific `location`. If no treasure is found at the location, return -1.
4. **Method 3: RemoveTreasure**: Implement a method `void RemoveTreasure(int location)`, which removes the treasure at the given `location`.
5. **Method 4: TotalTreasureValue**: Implement a method `int TotalTreasureValue()`, which returns the sum of all the treasure values in the map.

Write a C++ class `TreasureMap` with these methods. Ensure that you encapsulate the treasure data and handle any necessary data structures and constraints.

**Input:**

- The first line contains an integer `n` (1 <= n <= 100), the number of treasure locations.

- The following `n` lines contain two integers each: `location` and `value`, representing the location and value of each treasure.
-

**Output:**

For each operation of type `2`, `3`, or `4`, there is no output.

For each operation of type `1`, there is no output.

For each operation of type `5`, output the total treasure value.

**Constraints:**

- The `location` of a treasure is unique and will be an integer between 1 and 10^6.
- The `value` of a treasure will be an integer between 1 and 10^3.
- The sum of all treasure values does not exceed 10^6.

Here are three test cases to check the solution:

**Test Case 1:**

**Input:**

```
4
1 100
2 50
3 200
4 150
5
```

**Output:**

```
200
```

**Test Case 2:**

**Input:**

```
5
1 100
2 50
3 200
4 150
4 300
5
```

### Output:

```
400
```

### Test Case 3:

### Input:

```
3
1 10
1 20
1 30
2
5
```

### Output:

```
60
```

# 3. Problem Title: Abstract Shapes

**Problem Statement:**

You are required to design a C++ class hierarchy to represent abstract shapes. The class hierarchy should include the following shapes:

1. **Circle**: Represented by a center point (x, y) and a radius.
2. **Rectangle**: Represented by the coordinates of the top-left and bottom-right corners.
3. **Triangle**: Represented by the coordinates of its three vertices.

You need to create a base class `Shape` and derive the above three shapes from it. The `Shape` class should define a pure virtual function called `Area()` that will be implemented in each of the derived shape classes.

Your task is to design the class hierarchy and provide the implementation for each shape's `Area()` method.

**Input:**

- The first line contains an integer `q` (1 <= q <= 100), the number of queries.
- The following `q` lines each represent a query. Each query has the following format:
  - For a `Circle`: "1 x y r" (1 <= x, y <= 1000, 1 <= r <= 1000)
  - For a `Rectangle`: "2 x1 y1 x2 y2" (1 <= x1, y1, x2, y2 <= 1000, x1 < x2, y1 > y2)
  - For a `Triangle`: "3 x1 y1 x2 y2 x3 y3" (1 <= x1, y1, x2, y2, x3, y3 <= 1000)

**Output:**

For each query, output the area of the corresponding shape, rounded to two decimal places.

**Constraints:**

- All floating-point values will have at most two decimal places.

Here are three test cases to check the solution:

### Test Case 1:

### Input:

```
4
1 0 0 5
2 1 3 5 1
3 1 1 3 4 5 1
1 2 2 4
```

### Output:

```
78.54
8.00
6.00
12.57
```

### Test Case 2:

### Input:

```
2
2 1 1 5 4
3 1 2 4 2 3 5
```

### Output:

```
15.00
6.50
```

### Test Case 3:

### Input:

```
1
1 10 10 2
```

### Output:

```
12.57
```

# 4. Problem: Find Maximum and Minimum Element Using Pointers

You are given an array of integers, and your task is to find the maximum and minimum elements in the array using C++ pointers.

Write a C++ function with the following signature:

```
void find_max_min(int *arr, int size, int *max, int *min);
```

**Input Format:**

- The first line contains an integer `N` (1 ≤ N ≤ 100), the number of elements in the array.
- The next line contains `N` space-separated integers, the elements of the array.

**Output Format:**

- The maximum and minimum elements are printed on separate lines.

**Sample Input:**

```
5
12 45 7 89 23
```

**Sample Output:**

```
Maximum Element: 89
Minimum Element: 7
```

**Test Cases:**

Test Case 1:

**Input:**

```
4
10 20 30 40
```

**Output:**

```
Maximum Element: 40
Minimum Element: 10
```

Test Case 2:

**Input:**

```
1
100
```

**Output:**

```
Maximum Element: 100
Minimum Element: 100
```

Test Case 3:

**Input:**

```
6
-5 -10 -15 -20 -25 -30
```

**Output:**

```
Maximum Element: -5
Minimum Element: -30
```

# 5. Problem Title: Secret Vault

**Problem Statement:**

You are tasked with creating a secret vault that stores valuable items. The vault has a limited storage capacity. Your job is to design a C++ class `Vault` that hides the data related to the vault's contents and provides methods for interacting with the vault. The class should have the following functionalities:

1. **Initialization**: The class should be initialized with an integer `capacity`, representing the maximum number of items the vault can hold.
2. **Method 1: AddItem**: Implement a method `void AddItem(string item)`, which adds an item to the vault. If the vault is full, don't add the item.
3. **Method 2: RemoveItem**: Implement a method `void RemoveItem(string item)`, which removes an item from the vault if it exists.
4. **Method 3: ContainsItem**: Implement a method `bool ContainsItem(string item)`, which checks if the vault contains a specific item and returns true if the item is found, and false otherwise.
5. **Method 4: IsFull**: Implement a method `bool IsFull()`, which returns true if the vault is full and false if there is space for more items.

Write a C++ class `Vault` with these methods and ensure that the vault's data is hidden from direct access.

**Input:**

- The first line contains an integer `capacity` (1 <= capacity <= 100), the maximum number of items the vault can hold.
- The second line contains an integer `q` (1 <= q <= 100), the number of queries.
- The following `q` lines represent the queries. Each query has one of the following formats:

- "1 item" (Add the item to the vault)
- "2 item" (Remove the item from the vault)
- "3 item" (Check if the item is in the vault)
- "4" (Check if the vault is full)

**Output:**

For each query of type `3`, if the item is found in the vault, output "1." Otherwise, output "0."

For each query of type `4`, if the vault is full, output "1." Otherwise, output "0."

For other query types, there is no output.

**Constraints:**

- The `item` in each query is a string of alphanumeric characters, and its length is at most 20 characters.

Here are three test cases to check the solution:

**Test Case 1:**

**Input:**

```
3
6
1 GoldBars
1 Diamonds
2 GoldBars
1 SilverCoins
3 Diamonds
4
```

**Output:**

```
0
1
```

```
1
```

### Test Case 2:

### Input:

```
2
7
1 Passport
2 CreditCard
1 Jewels
4
3 Passport
1 Money
3 Money
4
```

### Output:

```
1
1
0
0
```

### Test Case 3:

### Input:

```
1
5
1 SecretDocument
1 AncientScroll
2 AncientScroll
3 SecretDocument
4
```

### Output:

```
0
1
```

```
0
```

# 6 Problem Title: Inheritance Hierarchy

**Problem Statement:**

You are tasked with designing a C++ class hierarchy representing vehicles. The class hierarchy should include a base class `Vehicle` and two derived classes, `Car` and `Bicycle`, to demonstrate inheritance. The base class `Vehicle` should contain common attributes and methods that are shared by all vehicles, and the derived classes should have specific attributes and methods that are unique to each type of vehicle.

Here are the details of the classes:

1. **Vehicle Class (Base Class):**
    - Attributes:
        - An integer `speed` representing the speed of the vehicle in kilometers per hour (1 <= speed <= 100).
        - An integer `wheels` representing the number of wheels of the vehicle (1 for bicycles, 4 for cars).
    - Methods:
        - A constructor to initialize the speed and wheels.
        - `void Accelerate(int increment)`: Increases the speed of the vehicle by the given increment.
        - `void Brake(int decrement)`: Decreases the speed of the vehicle by the given decrement.
        - `int GetSpeed() const`: Returns the current speed of the vehicle.
        - `int GetWheels() const`: Returns the number of wheels of the vehicle.

2. **Car Class (Derived Class):**
    - Attributes:
        - An integer `fuel` representing the amount of fuel in liters (1 <= fuel <= 50).
    - Methods:
        - A constructor to initialize the fuel and call the base class constructor.
        - `void Refuel(int amount)`: Increases the fuel level by the given amount.
        - `int GetFuel() const`: Returns the current fuel level.
3. **Bicycle Class (Derived Class):**
    - No additional attributes or methods, as bicycles do not have fuel.

Write the C++ classes and demonstrate how to use them. The main program should create instances of `Car` and `Bicycle`, perform some operations (e.g., accelerating, braking, refueling), and output the results.

**Input:**

- No specific input is required. You can demonstrate the usage of the classes with sample operations in the main program.

**Output:**

- Output the results of operations (e.g., current speed, current fuel level) based on the interactions with the created `Car` and `Bicycle` objects.

# 7 Problem Title: Polymorphic Zoo

**Problem Statement:**

You are tasked with creating a program to manage a zoo with different types of animals. The animals in the zoo include birds and mammals. Each type of animal has specific attributes and behaviors. You need to design a C++ class hierarchy to represent these animals and demonstrate polymorphism.

Here are the details of the classes:

1. **Animal Class (Base Class):**
    - Attributes:
        - A string `name` representing the name of the animal (length <= 20).
        - An integer `age` representing the age of the animal (1 <= age <= 100).
    - A constructor to initialize the `name` and `age`.
    - A pure virtual method `void MakeSound()` that represents the sound the animal makes.
2. **Bird Class (Derived Class):**
    - Attributes:
        - A string `species` representing the species of the bird (e.g., "Eagle").
    - A constructor to initialize the `name`, `age`, and `species`.
    - Implement the `MakeSound()` method to make the bird sound like "Chirp!"
3. **Mammal Class (Derived Class):**
    - Attributes:
        - A string `species` representing the species of the mammal (e.g., "Lion").
    - A constructor to initialize the `name`, `age`, and `species`.

- Implement the `MakeSound()` method to make the mammal sound like "Roar!"

Write the C++ classes and demonstrate polymorphism by creating an array of pointers to the base class `Animal` and adding instances of `Bird` and `Mammal` to the array. Use a loop to iterate through the array and call the `MakeSound()` method for each animal. The program should output the name, age, species, and the sound of each animal.

**Input:**

- No specific input is required. You can demonstrate the usage of the classes with sample operations in the main program.

**Output:**

- Output the information (name, age, species, sound) for each animal in the zoo.

t**est Case 1:**

**Input:**

No specific input is required; the program creates instances of animals in the main function.

**Output:**

```
Name: Sparrow, Age: 2, Species: Sparrow, Sound: Chirp!
Name: Parrot, Age: 5, Species: Parrot, Sound: Chirp!
Name: Lion, Age: 8, Species: Lion, Sound: Roar!
Name: Eagle, Age: 10, Species: Eagle, Sound: Chirp!
Name: Tiger, Age: 6, Species: Tiger, Sound: Roar!
```

# 8 Problem Title: Message Relay

**Problem Statement:**

You are given a set of entities, each represented by a unique integer ID from 1 to N. These entities are connected by a network of message-passing links. Each link is directed and connects two entities. The entities can send messages to each other through these links.

Your task is to simulate a message-passing system, where you'll process a sequence of messages and determine which entity receives each message.

Write a C++ program that performs the following operations:

1. **Initialization:** Initialize the system with N entities, and M directed links connecting these entities.
2. **Method 1: Send Message:** Implement a method `void SendMessage(int sender, int receiver, string message)` that sends a message from one entity to another. The message is sent from the sender to the receiver.
3. **Method 2: Receive Messages:** Implement a method `vector<string> ReceiveMessages(int entity)` that returns a list of messages received by a given entity.
4. **Method 3: Broadcast Message:** Implement a method `void BroadcastMessage(int sender, string message)` that sends a message from one entity to all entities connected to it through outgoing links.

Write the C++ class `MessageRelay` with these methods. Ensure that the system correctly processes and delivers messages. The messages should be delivered based on the order in which they are sent.

**Input:**

- The first line contains two integers, N and M, representing the number of entities (1 <= N <= 1000) and the number of links (1 <= M <= N * (N - 1)).

- The following M lines contain two integers, `sender` and `receiver`, representing the source and target entities of the directed links. These links are 0-indexed.
- The following line contains an integer Q (1 <= Q <= 1000), representing the number of queries.
- The following Q lines represent the queries. Each query has one of the following formats:
  - "1 sender receiver message" (Send a message from sender to receiver with the given message.)
  - "2 entity" (Receive messages for the given entity.)
  - "3 sender message" (Broadcast a message from sender to all connected entities with the given message.)

**Output:**

- For each query of type 2, output the received messages for the entity in the order they were received.

Here are two test cases to check the solution:

**Test Case 1:**

**Input:**

```
4 4
0 1
1 2
1 3
3 0
6
1 0 2 Hello
2 2
1 1 3 Hi
1 2 1 Hey
3 0 Bye
2 0
```

**Output:**

```
Hi
```

### Test Case 2:

### Input:

```
3 2
0 1
1 2
5
1 0 1 Hello
2 0
1 2 1 Hey
3 0 Hi
2 2
```

### Output:

```
Hello
Hey
Hi
```

# 9 Problem Title: Nested Shapes

**Problem Statement:**

You are tasked with designing a C++ program to work with geometric shapes. The program should have two classes: `Shape` and `ShapeContainer`.

1. **Shape Class (Outer Class):**
    - The `Shape` class represents a geometric shape and has the following methods:
        - A constructor that takes an integer `shapeID` (1 <= shapeID <= 100) and initializes the shape's ID.
        - A method `GetID()` that returns the shape's ID.
        - A pure virtual method `double GetArea()` that calculates and returns the area of the shape. This method will be implemented in derived classes.
2. **ShapeContainer Class (Nested Class):**
    - The `ShapeContainer` class is a nested class within the `Shape` class. It is responsible for storing and managing a collection of shapes. The `ShapeContainer` class should have the following methods:
        - A constructor that initializes an empty collection of shapes.
        - A method `AddShape(Shape* shape)` that adds a shape to the collection.
        - A method `GetTotalArea()` that calculates and returns the total area of all shapes in the collection.

Your task is to implement the `Shape` and `ShapeContainer` classes and demonstrate their usage in the `main` function.

**Input:**

- The input consists of two parts:

    - The first part contains a positive integer `N` (1 <= N <= 10), the number of shapes.
    - The following `N` lines describe the shapes. Each shape is defined as a single line consisting of two parts separated by a space:
        - An integer `shapeID` (1 <= shapeID <= 100) representing the shape's ID.
        - A string that specifies the shape type, which is one of the following: "Circle," "Rectangle," or "Triangle."
- The second part contains an integer `M` (1 <= M <= 10), the number of queries.
- The following `M` lines are queries. Each query consists of a single integer `queryType` (1 or 2):
    - Type 1: Add a shape to the container. In this case, the query line contains a shape ID (1 <= shapeID <= 100) followed by the shape type.
    - Type 2: Calculate and print the total area of all shapes in the container.
    -

**Output:**

- For each query of type 2, output the total area of all shapes in the container rounded to two decimal places.

**Example:**

**Input:**

```
3
1 Circle
2 Rectangle
3 Triangle
2
1 4 Rectangle
2
```

**Output:**

```
0.00
12.57
```

**Explanation:**

- Three shapes with IDs 1, 2, and 3 are initially provided (Circle, Rectangle, and Triangle).
- Query 1 adds a Rectangle with ID 4 to the container.
- Query 2 calculates and prints the total area of all shapes in the container.

# 10 Problem Title: Variable Comparison

**Problem Statement:**

You are provided with a set of variable assignments, each involving pointer and reference variables. Your task is to determine the final values of the variables after all assignments are applied.

The variables are integers, and you need to implement a C++ program to simulate the assignments. There are two types of assignments:

1. **Pointer Assignment:** You are given an integer value and a pointer variable. The value is assigned to the memory location pointed to by the pointer.
2. **Reference Assignment:** You are given two reference variables. The value of the first reference variable is assigned to the second reference variable.

Write a C++ program that performs these assignments and outputs the final values of the variables.

**Input:**

- The first line contains an integer N (1 <= N <= 100), the number of assignments.
- The following N lines describe the assignments. Each assignment is represented as one of the following:
  - Pointer Assignment: "p value" where value is an integer (1 <= value <= 10^6).
  - Reference Assignment: "r1 r2" where r1 and r2 are integers representing reference variables.
  -

**Output:**

- Output the final values of all variables after performing all assignments, separated by spaces.

**Example:**

**Input:**

```
5
p 3
r1 4
p 7
r2 5
p 2
```

**Output:**

```
2 5 7
```

**Explanation:**

- Initial variables: r1=4, r2=5, p=0
- After the first pointer assignment: r1=4, r2=5, p=3
- After the first reference assignment: r1=4, r2=4, p=3
- After the second pointer assignment: r1=4, r2=4, p=7
- After the second reference assignment: r1=7, r2=7, p=7
- After the third pointer assignment: r1=7, r2=7, p=2
- Final variables: r1=7, r2=7, p=2

# 11 Problem Title: Geometry Shapes

**Problem Statement:**

You are tasked with implementing a C++ program to work with geometric shapes, including circles and rectangles. Each shape has specific attributes and methods. Your task is to design classes to represent these shapes, along with constructors for initializing them.

1. **Shape Class (Base Class):**
   - The `Shape` class is the base class for all geometric shapes and has the following methods:
     - A constructor to initialize the shape's name (a string), which will be at most 20 characters long.
     - A pure virtual method `double GetArea()` that calculates and returns the area of the shape. This method will be implemented in derived classes.
2. **Circle Class (Derived Class):**
   - The `Circle` class is derived from the `Shape` class and has the following attributes and methods:
     - A constructor that takes the shape's name and radius (a positive floating-point number).
     - Implement the `GetArea()` method to calculate the area of a circle as π * radius^2.
3. **Rectangle Class (Derived Class):**
   - The `Rectangle` class is derived from the `Shape` class and has the following attributes and methods:
     - A constructor that takes the shape's name, width, and height (positive floating-point numbers).

- Implement the `GetArea()` method to calculate the area of a rectangle as width * height.

Write the C++ classes for `Shape`, `Circle`, and `Rectangle`, including their constructors. Demonstrate their usage in the `main` function.

**Input:**

- The input consists of three lines:
  - The first line contains the shape's name as a string (up to 20 characters).
  - The second line contains the string "Circle" or "Rectangle" to specify the type of shape.
  - The third line contains the attributes for the shape as follows:
    - For a "Circle," a single floating-point number (radius).
    - For a "Rectangle," two floating-point numbers (width and height).
    -

**Output:**

- Output the area of the shape with two decimal places.
-

**Example:**

**Input:**

```
MyCircle
Circle
3.5
```

**Output:**

```
38.48
```

**Explanation:**

- We create a circle with a radius of 3.5 units and a name "MyCircle."
- The area of the circle is calculated as π * 3.5^2 ≈ 38.48.

# 12 Problem Title: Resource Cleanup

**Problem Statement:**

You are tasked with implementing a C++ program that involves managing resources through constructors and destructors. Specifically, you are working with a collection of resources, represented by the `Resource` class. Your task is to design the class and its destructor to ensure proper resource cleanup.

The `Resource` class has two methods:

1. A constructor that initializes the resource with an ID (an integer from 1 to 1000).
2. A destructor that cleans up the resource.

Write a C++ program to create a collection of `Resource` objects and demonstrate the order in which the destructors are called when the program ends. Your program should output the IDs of the resources in the order in which their destructors are called.

**Input:**

- The input consists of a single integer `N` (1 <= N <= 1000), representing the number of `Resource` objects to create.
-

**Output:**

- Output the IDs of the `Resource` objects in the order in which their destructors are called, separated by spaces.

**Example:**

**Input:**

```
5
```

**Output:**

```
5 4 3 2 1
```

**Explanation:**

- We create five `Resource` objects with IDs from 1 to 5.
- The program ends, and the destructors are called in reverse order, cleaning up the resources and outputting their IDs in reverse order.

**Test Case 1:**

**Input:**

```
3
```

**Output:**

```
3 2 1
```

**Test Case 2:**

**Input:**

```
7
```

**Output:**

```
7 6 5 4 3 2 1
```

**Test Case 3:**

### Input:

```
1
```

### Output:

```
1
```

# 13 Problem Title: Complex Numbers

**Problem Statement:**

You are tasked with designing a C++ program to work with complex numbers. Complex numbers are represented as a sum of a real part and an imaginary part, where the imaginary part is multiplied by 'i' (the imaginary unit).

Your task is to implement a `Complex` class with the following functionalities:

1. **Constructor:** Create a constructor that initializes a complex number using two real numbers, `real` and `imaginary`.
2. **Overloaded Operators:**
   - Implement the addition `+` operator to add two complex numbers.
   - Implement the subtraction `-` operator to subtract two complex numbers.
   - Implement the multiplication `*` operator to multiply two complex numbers.
   - Implement the division `/` operator to divide two complex numbers.
3. **Print Method:** Implement a method `void Print()` that prints the complex number in the following format: `(real + imaginary * i)`.

Write a C++ program to create `Complex` objects, perform operations, and print the results.

**Input:**

- The input consists of two lines:
  1. The first line contains two real numbers, `real1` and `imaginary1`, separated by a space (|real1|, |imaginary1| <= 1000).
  2. The second line contains two real numbers, `real2` and `imaginary2`, separated by a space (|real2|, |imaginary2| <= 1000).

3.

**Output:**

- For each operation (addition, subtraction, multiplication, and division), print the result as a complex number in the specified format.
-

**Example:**

**Input:**

```
2 3
-1 2
```

**Output:**

```
(1 + 5 * i)
(3 + 1 * i)
(-8 + 1 * i)
(-0.384615 + 0.923077 * i)
```

**Explanation:**

- Two complex numbers are provided: `2 + 3i` and `1 + 2i`.
- The program performs addition, subtraction, multiplication, and division operations on these numbers and prints the results in the specified format.

# 14 Problem Title: Social Network

**Problem Statement:**

You are tasked with implementing a C++ program to model a social network. You have two classes: `User` and `SocialNetwork`. The `User` class represents a user on the social network, and the `SocialNetwork` class is responsible for managing the network.

1. **User Class:**
   - The `User` class represents a user on the social network and has the following attributes and methods:
     - A private integer `userID` (1 <= userID <= 1000) to identify the user.
     - A private string `username` (up to 20 characters) for the user's username.
     - A private set of integers `friends` to store the user's friend IDs.
     - A constructor that initializes the `userID` and `username`.
     - A method `void AddFriend(int friendID)` to add a friend by their ID to the user's friend list.
     - A method `void PrintFriends()` to print the usernames of the user's friends.
2. **SocialNetwork Class:**
   - The `SocialNetwork` class manages the social network and has the following attributes and methods:
     - A private vector of `User` objects to store user information.
     - A constructor that initializes the network.
     - A method `void CreateUser(int userID, const string& username)` to create a new user with the given `userID` and `username`.
     - A friend function `void MakeFriends(int userID1, int userID2)` that adds user `userID2` to the friend list of user `userID1`.
     - A friend function `void PrintUserFriends(int userID)` that prints the usernames of user `userID`'s friends.

Write a C++ program to create users, make friends, and print the friends of a user.

**Input:**

- The input consists of two parts:
  1. The first part contains an integer `N` (1 <= N <= 1000), representing the number of users.
  2. The following `N` lines describe the users, each with an integer `userID`, a string `username`, and an integer `M` (1 <= M <= N) representing the number of friends for that user.
  3. The next `M` lines list the user IDs of the friends for each user.
  4. 

**Output:**

- For each user, print their username and the usernames of their friends.

**Example:**

**Input:**

```
3
1 Alice 2
2 3
2 Bob 1
3
3 Carol 1
2
```

**Output:**

```
Alice: Bob Carol
Bob: Alice Carol
Carol: Alice Bob
```

**Explanation:**

- Three users are created with the names Alice, Bob, and Carol.
- User 1 (Alice) has two friends: User 2 (Bob) and User 3 (Carol).
- User 2 (Bob) has three friends: User 1 (Alice), User 2 (Bob), and User 3 (Carol).
- User 3 (Carol) has two friends: User 1 (Alice) and User 2 (Bob).
- The program prints the usernames of each user along with their friends' usernames.

# 15 Problem Title: Animal Kingdom

**Problem Statement:**

You are tasked with implementing a C++ program to model the animal kingdom. You have a base class `Animal` and three derived classes: `Mammal`, `Bird`, and `Fish`. Each class represents a group of animals with specific attributes and methods.

1. **Animal Class (Base Class):**
   - The `Animal` class is the base class for all animals and has the following attributes and methods:
     - A protected string `name` (up to 50 characters) to represent the name of the animal.
     - A constructor to initialize the `name`.
     - A pure virtual method `void Sound()` that represents the sound the animal makes. This method will be implemented in derived classes.
2. **Mammal Class (Derived Class):**
   - The `Mammal` class is derived from the `Animal` class and has the following attributes and methods:
     - A constructor that initializes the `name`.
     - Implement the `Sound()` method to print "Mammal sound."
3. **Bird Class (Derived Class):**
   - The `Bird` class is derived from the `Animal` class and has the following attributes and methods:
     - A constructor that initializes the `name`.
     - Implement the `Sound()` method to print "Bird sound."
4. **Fish Class (Derived Class):**

- The `Fish` class is derived from the `Animal` class and has the following attributes and methods:
  - A constructor that initializes the `name`.
  - Implement the `Sound()` method to print "Fish sound."

Write a C++ program to create objects of the `Mammal`, `Bird`, and `Fish` classes, and print the sounds they make.

**Input:**

- The input consists of a single integer `N` (1 <= N <= 100), representing the number of animals.
- The following `N` lines describe the animals, each with an integer `type` (1 for `Mammal`, 2 for `Bird`, 3 for `Fish`) and a string `name` (up to 50 characters).

**Output:**

- For each animal, print its name and the sound it makes.

**Example:**

**Input:**

```
3
1 Lion
2 Eagle
3 Salmon
```

**Output:**

```
Lion makes Mammal sound.
Eagle makes Bird sound.
Salmon makes Fish sound.
```

**Explanation:**

- Three animals are created with their respective types and names.
- The program calls the `Sound()` method for each animal to print their names and the sounds they make.

# 16 Problem Title: Shapes and Areas

**Problem Statement:**

You are tasked with implementing a C++ program to work with geometric shapes, including circles and rectangles. Each shape has specific attributes and methods. Your task is to design classes to represent these shapes using abstract classes and calculate their areas.

1. **Abstract Shape Class:**
   - Create an abstract class `Shape` with the following attributes and methods:
     - A pure virtual method `double GetArea()` that calculates and returns the area of the shape. This method will be implemented in derived classes.
2. **Circle Class (Derived from Shape):**
   - The `Circle` class is derived from the `Shape` class and has the following attributes and methods:
     - A constructor that initializes the radius (a positive floating-point number).
     - Implement the `GetArea()` method to calculate the area of a circle as π * radius^2.
3. **Rectangle Class (Derived from Shape):**
   - The `Rectangle` class is derived from the `Shape` class and has the following attributes and methods:
     - A constructor that initializes the width and height (positive floating-point numbers).
     - Implement the `GetArea()` method to calculate the area of a rectangle as width * height.

Write a C++ program to create objects of the `Circle` and `Rectangle` classes, calculate their areas, and print the areas.

**Input:**

- The input consists of three lines:
  1. The first line contains a floating-point number `radius` (1 <= radius <= 1000) for the circle.
  2. The second line contains two floating-point numbers `width` and `height` (1 <= width, height <= 1000) for the rectangle.

**Output:**

- Output the area of the circle and the area of the rectangle, each with two decimal places, separated by a space.

**Example:**

**Input:**

```
4.5
3 5
```

**Output:**

```
63.62 15.00
```

**Explanation:**

- We create a circle with a radius of 4.5 units and a rectangle with dimensions 3x5 units.
- The area of the circle is calculated as π * 4.5^2 ≈ 63.62, and the area of the rectangle is 3 * 5 = 15.00.

# 17 Problem Title: Animal Sounds

**Problem Statement:**

You are tasked with implementing a C++ program to model the animal kingdom, focusing on polymorphism. You have a base class `Animal` and three derived classes: `Mammal`, `Bird`, and `Fish`. Each class represents a group of animals with specific attributes and methods.

1. **Animal Class (Base Class):**
    - The `Animal` class is the base class for all animals and has the following attributes and methods:
        - A private string `name` (up to 50 characters) to represent the name of the animal.
        - A constructor to initialize the `name`.
        - A method `void Sound()` that prints "Animal sound."
2. **Mammal Class (Derived Class):**
    - The `Mammal` class is derived from the `Animal` class and has the following attributes and methods:
        - A constructor that initializes the `name`.
        - Implement the `Sound()` method to print "Mammal sound."
3. **Bird Class (Derived Class):**
    - The `Bird` class is derived from the `Animal` class and has the following attributes and methods:
        - A constructor that initializes the `name`.
        - Implement the `Sound()` method to print "Bird sound."
4. **Fish Class (Derived Class):**
    - The `Fish` class is derived from the `Animal` class and has the following attributes and methods:

- A constructor that initializes the `name`.
- Implement the `Sound()` method to print "Fish sound."

Write a C++ program to create objects of the `Mammal`, `Bird`, and `Fish` classes, and print the sounds they make.

**Input:**

- The input consists of a single integer `N` (1 <= N <= 100), representing the number of animals.
- The following `N` lines describe the animals, each with an integer `type` (1 for `Mammal`, 2 for `Bird`, 3 for `Fish`) and a string `name` (up to 50 characters).

**Output:**

- For each animal, print its name and the sound it makes.

**Example:**

**Input:**

```
3
1 Lion
2 Eagle
3 Salmon
```

**Output:**

```
Lion makes Mammal sound.
Eagle makes Bird sound.
Salmon makes Fish sound.
```

**Explanation:**

- Three animals are created with their respective types and names.

- The program calls the `Sound()` method for each animal to print their names and the sounds they make.

# 18 Problem Title: Shape Calculator

**Problem Statement:**

You are tasked with implementing a C++ program for a shape calculator. The program should be able to calculate the area and perimeter of different geometric shapes, including circles and rectangles. Your task is to design classes to represent these shapes and use virtual functions to calculate their areas and perimeters.

1. **Shape Class (Base Class):**
   - Create a base class `Shape` with the following attributes and methods:
     - A protected string `name` (up to 20 characters) to represent the name of the shape.
     - A constructor to initialize the `name`.
     - A pure virtual method `double GetArea()` that calculates and returns the area of the shape.
     - A pure virtual method `double GetPerimeter()` that calculates and returns the perimeter of the shape.
2. **Circle Class (Derived from Shape):**
   - The `Circle` class is derived from the `Shape` class and has the following attributes and methods:
     - A protected floating-point number `radius` (a positive number) to represent the radius of the circle.
     - A constructor to initialize the `name` and `radius`.
     - Implement the `GetArea()` method to calculate the area of a circle as π * radius^2.
     - Implement the `GetPerimeter()` method to calculate the perimeter of a circle as 2 * π * radius.

3. **Rectangle Class (Derived from Shape):**
    - The `Rectangle` class is derived from the `Shape` class and has the following attributes and methods:
        - A protected floating-point numbers `width` and `height` (both positive numbers) to represent the dimensions of the rectangle.
        - A constructor to initialize the `name`, `width`, and `height`.
        - Implement the `GetArea()` method to calculate the area of a rectangle as width * height.
        - Implement the `GetPerimeter()` method to calculate the perimeter of a rectangle as 2 * (width + height).

Write a C++ program to create objects of the `Circle` and `Rectangle` classes, calculate their areas and perimeters, and print the results.

**Input:**

- The input consists of two lines:
    1. The first line contains a string `shapeName` (up to 20 characters) representing the name of the shape.
    2. The second line contains an integer `shapeType` (1 for `Circle`, 2 for `Rectangle`).
    3. If `shapeType` is 1 (Circle), the third line contains a floating-point number `radius`.
    4. If `shapeType` is 2 (Rectangle), the fourth line contains two floating-point numbers `width` and `height`.

**Output:**

- Output the calculated area and perimeter of the shape, each with two decimal places, separated by a space.

**Example:**

**Input:**

```
CircleShape
1
4.5
```

**Output:**

```
Area: 63.62 Perimeter: 28.27
```

**Explanation:**

- A circle named "CircleShape" is created with a radius of 4.5 units.
- The area of the circle is calculated as π * 4.5^2 ≈ 63.62, and the perimeter is calculated as 2 * π * 4.5 ≈ 28.27.

# 19 Problem Title: Vehicle Rental System

**Problem Statement:**

You are tasked with implementing a C++ program for a vehicle rental system. The system should allow customers to rent different types of vehicles, including cars and bikes. Each vehicle has specific attributes and methods. Your task is to design classes to represent these vehicles and use dynamic binding to calculate rental costs.

1. **Vehicle Class (Base Class):**
   - Create a base class `Vehicle` with the following attributes and methods:
     - A protected string `model` (up to 50 characters) to represent the model of the vehicle.
     - A protected integer `year` (4 digits) to represent the year of manufacture.
     - A constructor to initialize the `model` and `year`.
     - A pure virtual method `double CalculateRentalCost(int days)` that calculates and returns the rental cost for the vehicle for a given number of rental days.
2. **Car Class (Derived from Vehicle):**
   - The `Car` class is derived from the `Vehicle` class and has the following attributes and methods:
     - A protected boolean `isLuxury` to represent whether the car is a luxury car or not.
     - A constructor to initialize the `model`, `year`, and `isLuxury`.
     - Implement the `CalculateRentalCost(int days)` method to calculate the rental cost for a car as follows:
       - For non-luxury cars, the rental cost is $40 per day.
       - For luxury cars, the rental cost is $80 per day.

3. **Bike Class (Derived from Vehicle):**
    - The `Bike` class is derived from the `Vehicle` class and has the following attributes and methods:
        - A constructor to initialize the `model` and `year`.
        - Implement the `CalculateRentalCost(int days)` method to calculate the rental cost for a bike as $10 per day.

Write a C++ program to create objects of the `Car` and `Bike` classes, calculate the rental costs for a given number of rental days, and print the results.

**Input:**

- The input consists of three lines:
    1. The first line contains a string `vehicleModel` (up to 50 characters) representing the model of the vehicle.
    2. The second line contains an integer `vehicleYear` (4 digits) representing the year of manufacture.
    3. The third line contains an integer `vehicleType` (1 for `Car`, 2 for `Bike`).
    4. If `vehicleType` is 1 (Car), the fourth line contains a boolean `isLuxury` (0 for non-luxury, 1 for luxury).
    5. The fifth line contains an integer `rentalDays` (1 <= rentalDays <= 30) representing the number of rental days.

**Output:**

- Output the rental cost for the vehicle, with two decimal places.

**Example:**

**Input:**

```
Sedan
2020
```

```
1
1
5
```

**Output:**

```
400.00
```

**Explanation:**

- A luxury car with model "Sedan" and year 2020 is created.
- The rental cost for 5 days is calculated as $80 per day, resulting in a total cost of $400.00.

# 20 Problem Title: Division with Exception Handling

**Problem Statement:**

You are tasked with implementing a C++ program to perform division operations with exception handling. The program should handle potential division by zero errors and report them appropriately. Your task is to design a program that performs division and handles exceptions.

1. **Division Function:**
   - Implement a function `double SafeDivision(double numerator, double denominator)` that takes two floating-point numbers `numerator` and `denominator`.
   - The function should perform the division `numerator / denominator`.
   - If the `denominator` is zero, the function should throw an exception of type `const char*` with the message "Division by zero is not allowed."
   - If the division is valid, return the result as a `double`.

Write a C++ program that uses the `SafeDivision` function to perform division operations and handles exceptions appropriately.

**Input:**

- The input consists of two lines:
  1. The first line contains a floating-point number `numerator` (-10^9 <= numerator <= 10^9).
  2. The second line contains a floating-point number `denominator` (-10^9 <= denominator <= 10^9).

**Output:**

- If the division is successful, output the result with two decimal places.
- If a division by zero error occurs, output "Division by zero is not allowed."

**Example:**

**Input:**

```
10
2
```

**Output:**

```
5.00
```

**Input:**

```
8
0
```

**Output:**

```
Division by zero is not allowed.
```

**Explanation:**

- In the first example, the division of 10 by 2 is valid, and the result is 5.00.
- In the second example, attempting to divide 8 by 0 results in a division by zero exception, and the program outputs the appropriate error message.