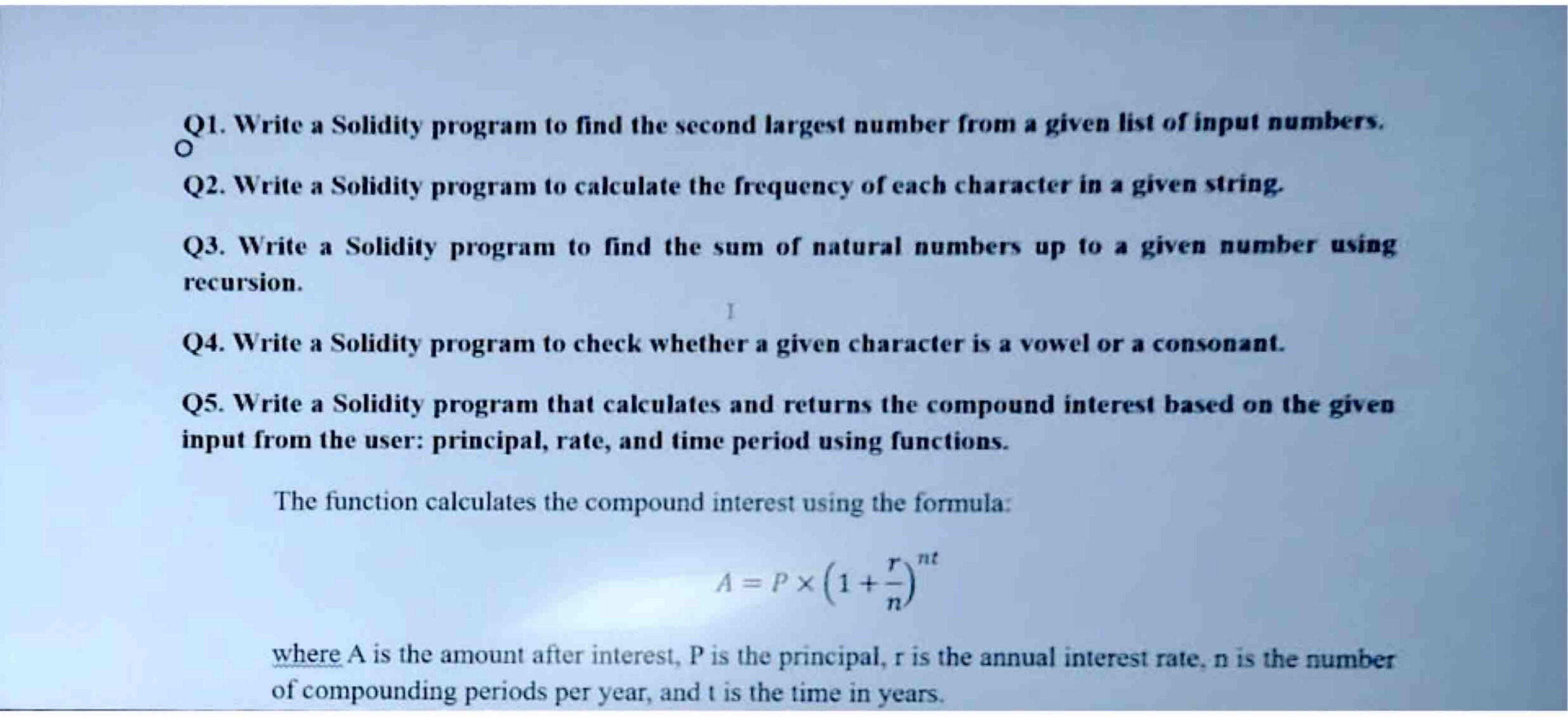

Q1. Write a Solidity program to find the second largest number from a given list of input numbers.

Q2. Write a Solidity program to calculate the frequency of each character in a given string.

Q3. Write a Solidity program to find the sum of natural numbers up to a given number using recursion.

Q4. Write a Solidity program to check whether a given character is a vowel or a consonant.

Q5. Write a Solidity program that calculates and returns the compound interest based on the given input from the user: principal, rate, and time period using functions.

The function calculates the compound interest using the formula:

$$A = P \times \left(1 + \frac{r}{n}\right)^{nt}$$

where A is the amount after interest, P is the principal, r is the annual interest rate, n is the number of compounding periods per year, and t is the time in years.

NOTE: Do not use any inbuilt methods

1. Write a method that takes an integer array as input and returns a new array with the elements in reverse order.
2. Create a method that accepts an array of integers and returns the maximum and minimum values found in the array.
3. Write a method that checks if there are any duplicate elements in an integer array. Return `true` if duplicates exist, otherwise return `false`.
4. Create a method that merges two sorted integer arrays into a single sorted array.
5. Write a method that rotates an array to the right by a given number of steps.
6. Write a method that checks whether a given string is a palindrome (reads the same forwards and backwards).
7. Create a method that counts the number of vowels and consonants in a given string.
8. Write a method that reverses the characters in a given string.
9. Create a method that returns the first non-repeated character in a string. If there are no non-repeated characters, return a special value (like `null`).
10. Write a method that checks whether two strings are anagrams of each other (contain the same characters in a different order).
11. Create a method that counts how many times a given substring appears in a string.

1. Write a class `Student` with fields `name` and `age`. Create a constructor that uses the `this` keyword to distinguish between instance variables and parameters.

2. Create a class `Rectangle` with two constructors: one that accepts both width and height, and another that only accepts width and sets a default height. Use the `this()` constructor chaining to avoid redundancy.

3. Write a class `Chain` where methods `step1()`, `step2()`, and `step3()` return `this` to allow method chaining.

4. Create a parent class `Animal` with a method `makeSound()` and a subclass `Dog` that overrides this method. Use the `super` keyword to call the parent class's method in the overridden version.

5. Write a class `Person` with a constructor that accepts `name` and `age`. Create a subclass `Employee` that accepts `name`, `age`, and `salary`, and use `super()` to initialize the name and age.

6. In the class `Vehicle` with a method `move()`, create a subclass `Car` that overrides `move()`. Use `super.move()` to call the superclass version inside the overridden method.

7. Write a class `Parent` with a method `display()`, and a subclass `Child` with a constructor that uses `super()` to call the parent class constructor. Use both `super.display()` and `this.display()` in the child class.

8. Create a class `Building` with overloaded constructors. Then create a class `House` that extends `Building`, and use `super()` to call different constructors from the superclass based on input parameters.

9. Create a class with an inner class and use `this` to refer to the outer class’s instance variables or methods.

10. Implement a `Human` class with two constructors. Create a `Student` class that calls both the `Hum an` constructors in different ways using `super()`.

1. Write a Java class `Student` that has multiple constructors. One constructor should accept student name and ID, while another should accept student name, ID, and grades. Implement constructor overloading and ensure the constructors call each other using `this()`.

2. Create a class `Employee` with fields `name`, `id`, `designation`, and `salary`. Implement multiple constructors that initialize different combinations of these fields, and ensure that they chain to a primary constructor using the `this()` keyword.

3. Design a class `ComplexNumber` that models complex numbers. Write a copy constructor that takes another `ComplexNumber` object and initializes the current object's real and imaginary parts with the copied values.

4. Implement a singleton class `DatabaseConnection` using a private constructor. Ensure that the class restricts object creation to only one instance and provides a global access point using a static method.

5. Write an abstract class `Shape` that has a parameterized constructor to initialize the color of the shape. Extend this class in `Circle` and `Rectangle`, which will have their own additional parameters (like radius, length, width). Ensure proper constructor calls using `super()`.

6. Implement a class `Polynomial` that models a polynomial equation. Use a constructor that takes a variable number of coefficients (using varargs) and initializes the polynomial. Write a method to display the polynomial in a readable format (e.g., `3x^2 + 2x + 1`).

7. Create an interface `Shape` that has methods `double area()` and `double perimeter()`. Implement this interface in two classes: `Circle` and `Rectangle`. The `Circle` class should calculate the area and perimeter using the radius, and the `Rectangle` class should use the length and width.

8. Create two interfaces `Flying` and `Swimming`, each with a method `void fly()` and `void swim()` respectively. Create a class `Duck` that implements both interfaces and overrides both methods. Write a test class that demonstrates the `Duck`'s ability to both fly and swim.

9. Create an interface `PaymentMethod` with a method `void pay(double amount)`. Implement this interface in two classes: `CreditCardPayment` and `PayPalPayment`. Write a class `OnlineStore` that accepts a `PaymentMethod` in its constructor and uses it to process a payment.