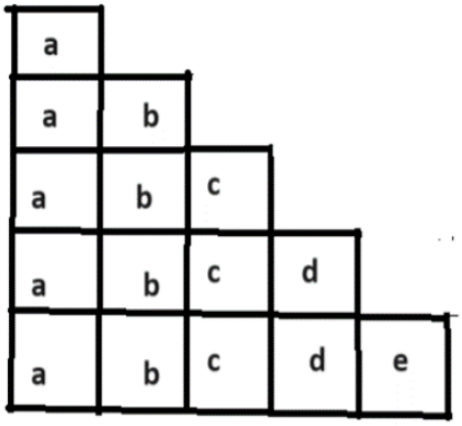
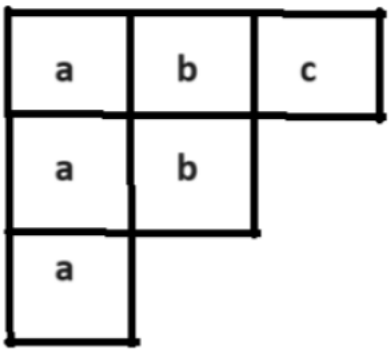
CSE215 section: 3	Assignment	Intersession 2023

1. Take an integer (n) and String (s) as user input and create a String array and fill them up with alphabets.
n will indicate the number of rows and s = (INC/DEC) will indicate whether it's an increasing array or a decreasing array.
For Example:

i. n = 5 and s = "INC", the array should look like this:



ii. n = 3 and s = "DEC", the array should look like this:



Question 2:

Quadratic Equation:

- A quadratic equation is an equation in the form $ax^2 + bx + c = 0$ where $a \neq 0$.
- For the quadratic function f(x) = a $(x + p)^2$ + q
- For the quadratic equation ax2 + bx + c = 0, the expression b2 – 4ac is called the discriminant. The value of the discriminant shows how many roots f(x) has:
  - If b2 – 4ac > 0 then the quadratic function has two distinct real roots.
  - If b2 – 4ac = 0 then the quadratic function has one repeated real root.
  - If b2 – 4ac < 0 then the quadratic function has no real roots.
- The equations for the finding one root of a Quadratic Equation **= (- b + SquareRoot(discriminant))/2a**
- The equations for the finding the other root of a Quadratic Equation **= (- b - SquareRoot(discriminant))/2a**

QuadraticEquation	**(12)**
- a : double - b : double - c : double	(1)
/** default & parameterized Constructors **/	(2)
/** setter getter methods **/	
+ FindDiscriminant():double	(1)
+ FindRoots( ) : double [ ]	(2)
+ toString() : String	(2)
+ SumOfDiscrimants(Qarray : QuadraticEquation[ ]) : double	(4)

In the "**QuadraticEquation**" class:

- **toString()** method: All the attributes values will be returned in a formatted way and also return the results of **FindDiscriminant():double** and **FindRoots( ) : double [ ]** methods.
- **FindRoots( )** method: if there is no real root, return -1.
- **SumOfDiscrimants(Qarray : QuadraticEquation[ ])** method: it will calculated the sum of discriminants of each elements of the Qarray which it takes as a parameter.

Now,

In the Tester (Main) Class: **(3)**

i. Take an integer user input n.
ii. Create an array of object of "**QuadraticEquation**" class of size n and fill them up with values.
iii. Call **SumOfDiscrimants(Qarray : QuadraticEquation[ ])** method.

Question 3:

Account
- accountNo : String - AccountName : Name - balance: double
+ parameterized Constructor + accessor & mutators + toString() : String

Name
- firstName : String - lastName: String
+ parameterized Constructor + accessor & mutators + toString() : String

Now,

In the Tester (Main) Class: **(3)**

i. Take an integer user input n.
ii. Create an array of object of "**Account**" class of size n and fill them up with user input values.
iii. Write a method called "WriteToFile" which will take a file as parameter and write to a file called "AccountRecords". This method will only write those objects of the array for which the accountNo is unique (means the accountNo doesn't already exist in the file).
(note: remember to write to the file in a tabular method with columns: AccountNO, FirstName, LastName, Balance)
iv. The program should throw a custom exception "DuplicateAccountException" whenever it encounters a duplicate account number or have negative number as balance.
v. (Bonus Question) Write another method called "ReadFile" which will take a file as parameter and read that file and throw a custom exception "CapitalException" whenever any Name has any Capital letter in it.

Question 4:

There is already a file called **"BookRecords"** in your system (inside your project folder) which holds the details of some Books and their Prices and how many copies Sold.

It holds the following details: (you don't need to write code for writing to the file)

BookName	WriterName	PublisherName	Price	Sold
Human	Ayon	ABC	10	105
Primal	Harry	RCD	18	22
Legend	Allen	Digital	7	55
Golden	Robin	ABC	11	35
Race	Harry	XYZ	15	215
Hacker	Robin	ABC	35	156
Animals	Ayon	Digital	12	22
Sparrow	Ayon	ABC	11	33
Vision	Robin	XYZ	17	72
Ruse	Maya	Banks	33	190

In The main method of Testing Class:

Now read the **"BookRecords"** file to find out:

(Note: Don't just use your own eyes to read the file.)

- Print The name of the most popular book in the file and who wrote it……..…..(3)
- Print how many **WriterName** (unique, excluding the duplicates) are there in the file?.....(6)
- Make a **Writer** class object for **WriterName** = "Ayon". And print his TotalEarning?.....(8)

So, What is TotalEarning = ??

Suppose, for one Writer, his/her total number of PublishedBooks = 3.
1st book price is = 10 and sold = 5 copies. So total selling for Book1 = 10x5 = 50.
2nd book price is = 5 and sold = 3 copies. So total selling for Book1 = 5x3 = 15.
3rd book price is = 100 and sold = 7 copies. So total selling for Book1 = 100x7 = 700.
So, for this writer, total selling = 50+15+700 = 765.
A writer gets 15% of total selling.
So, that writer's TotalEarning = 15% of 765

Answer any one of the following two:

Question 5:

Create a class named Pizza that stores information about a single pizza. It should contain the following:

- Private instance variables to store the size of the pizza (either small, medium, or large), the number of cheese toppings, the number of pepperoni toppings, and the number of ham toppings.

- Constructor(s) that set all of the instance variables.
- Public methods to get and set the instance variables.
- A public method named calcCost( ) that returns a double that is the cost of the pizza.
  Pizza cost is determined by:
  Small: $1tt + $2 per topping
  Medium: $12 + $2 per topping
  Large: $14 + $2 per topping
- public method named getDescription( ) that returns a String containing the pizza size, quantity of each topping.

Write test code to create several pizzas and output their descriptions. For example, a large pizza with one cheese, one pepperoni and two ham toppings should cost a total of $22. Now Create a PizzaOrder class that allows up to three pizzas to be saved in an order. Each pizza saved should be a Pizza object. Create a method calcTotal() that returns the cost of order. In the runner order two pizzas and return the total cost.

Question 6:

Create a SavingsAccount class. Use a static data member annualInterestRate to store the annual interest rate for each of the savers. Each member of the class contains a private data member savingsBalance indicating the amount the saver currently has on deposit. Provide member function calculateMonthlyInterest that calculates the monthly interest by multiplying the balance by annualInterestRate divided by 12; this interest should be added to savingsBalance. Provide a static member function modifyInterestRate that sets the static annualInterestRate to a new value.

Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of $2000.00 and $3000.00, respectively. Set annualInterestRate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month's interest and print the new balances for both savers.