NAME : HARSHITA AGARWAL

CLASS : XII

SECTION : S1

SUBJECT : COMPUTER SCIENCE

YEAR : 2024-2025

UID: 7792600

# INDEX


| Serial No. | Content | Page No. | T. Sign |
|---|---|---|---|
| 1 | Question 1 | | |
| 2 | Question 2 | | |
| 3 | Question 3 | | |
| 4 | Question 4 | | |
| 5 | Question 5 | | |
| 6 | Question 6 | | |
| 7 | Question 7 | | |
| 8 | Question 8 | | |
| 9 | Question 9 | | |
| 10 | Question 10 | | |
| 11 | Question 11 | | |
| 12 | Question 12 | | |
| 13 | Question 13 | | |
| 14 | Question 14 | | |
| 15 | Question 15 | | |
| 16 | Question 16 | | |
| 17 | Question 17 | | |
| 18 | Question 18 | | |
| 19 | Question 19 | | |
| 20 | Question 20 | | |
| 21 | Question 21 | | |
| 22 | Question 22 | | |
| 23 | Question 23 | | |
| 24 | Question 24 | | |
| 25 | Question 25 | | |

Q13) A set is a collection of elements in which there is no duplication of elements.

S = {1,6,9,24} is a set of 4 integer elements. An array of integers may be used to represent a set. You may assume that there will be a maximum of 50 elements in the set. Following are some member functions of the class set.

Data Members :-

arr[] : an array of integers

n : an integer to store the total number of elements.

Member Functions :-

Set (int nn) : Constructor to initialize n=nn and the array arr[]

void read() : reads the elements of the set

int getsize() : returns n. the number of elements in the set

int has(int ele) : returns 1 if ele belongs to the current set otherwise 0

Set intersection (Set d) : returns the intersection of set object d and the current set object

Set union (Set d) : returns the union of set object d and the current set object

Specify the class Set, giving details of the constructor and the functions. Also define a main() function to create an object and call the functions accordingly to enable the task.

## Algorithm :-

Step 1 : Start

Step 2 : Class starts

Step 3 : Initialize n and allocate memory for arr.

Step 4 : Use a method read to read n elements into arr.

Step 5 : Use a method show to print the elements of arr.

Step 6 : Use a method has to check if an element is in arr.

Step 7 : Create a method intersection to find common elements between two sets and store them in a new set.

Step 8 : Create a method union to combine elements from two sets without duplicates.

Step 9 : Main starts

Step 10 : Call methods

Step 11 : Main ends

Step 12 : Class ends

# Variable Description :-

| Sl.No | Variable Name | Datatype | Purpose |
|---|---|---|---|
| 1 | arr | int[] | Stores the elements of the set |
| 2 | n | int | Specifies the number of elements in the set |
| 3 | ele | int | Holds the element to be checked for presence in the set |
| 4 | result | Set | To store the result of the intersection or union of sets |
| 5 | k | int | Keeps track of the number of elements in the result set |

Q14) The Computer Department of the agency of International Espionage is trying to decode intercepted messages. The agency's spies have determined that the enemy encodes messages by first converting all characters to their ASCII values and then reversing the string.

For example consider A Z. The ASCII values of A Z are 65, 32 and 122 respectively. Concatenate them to get 6532122, then reverse this to get 2212356 as the coded message.

Write a program which reads a coded message and decodes it. The coded message will not exceed 200 characters. It will contain only alphabets (A..Z and a..z) and spaces. ASCII values of A…Z are 65 to 90 and those of a…z are 97 to 122. Test your program for the following data and some random data.

Algorithm :

Step 1 : Start class

Step 2 : Create a method decMes that takes encoMes as input

Step 3 : Reverse encoMes

Step 4 : Loop through the reversed dtring

Step 5 : Return decmes as a string

Step 6 : Main starts

Step 7 : Read encoded message from the user

Step 8 : Call decMes to decode

Step 9 : Print the decoded message

Step 10 : Main ends

Step 11 : Class ends

# Variable Description :-

| Sl.No | Variable Name | Datatype | Purpose |
|---|---|---|---|
| 1 | encoMes | String | Stores the input encoded message |
| 2 | revMes | String | Collects characters to form the decoded message |
| 3 | decMes | StringBuilder | Collects characters to form the decoded message |
| 4 | i | int | Tracks position within the reversed message |
| 5 | asciivalue | int | Stores the ASCII value extracted from the message |
| 6 | scanner | Scanner | Reads the encoded message from user |

Q15) A new advanced operating system, incorporating the latest hi-tech features has been designed by Opera Computer Systems. The task of generation copy protection codes to prevent software privacy has been entrusted to the security department. The security department has decided to have codes containing a jumbled combination of alternate upper-case letters of the alphabet starting from A up to K (namely among A,C,E,G,I,K). The code may or may not be in the consecutive series of alphabets. Each code should not exceed 6 characters, display an appropriate error message.

Write a program to input a code and its length. At the first instance of an error display “INVALID!” stating the appropriate reason. In case of no error, display the message “Valid!”.

# Algorithm :-

Step 1 : Class starts

Step 2 : Main starts

Step 3 : Create a scanner object for input

Step 4 : Prompt and read integer N

Step 5 : If N>6, print an error and exit

Step 6 : Prompt and read the code string CODE

Step 7 : If length of CODE != N, print an error and exit

Step 8 : Check each character of CODE :

Step 9 : If not uppercase, print an error and exit

Step 10 : If not A,E,C,G,I,K, print an error and exit.

Step 11 : Check for repeated characters, if found print an error and exit

Step 12 : If all checks pass, print “Valid!”

Step 13 : Main ends

Step 14 : Class ends

## Variable Description :-

| Sl.No | Variable Name | Datatype | Purpose |
|---|---|---|---|
| 1 | N | int | Stores the number of characters expected in the code string |
| 2 | i | int | Used to iterate through each character in the code string |
| 3 | j | int | Used to compare characters for duplication |
| 4 | CODE | String | Stores the user input code string |
| 5 | flag | int | Used to flag invalid conditions in the code string |
| 6 | l | int | Stores the length of the code string |
| 7 | z | char | Stores the current character being validated in the code string |

Q16) Design a program to accept a day number (between 1 and 366), year (in 4 digits) from the user to generate and display the corresponding date. Also accept N (1<=N<=100) from the user to compute and display the future date corresponding to N days after the generated day. Display an error message if the value of the day number, year and N are not within the limit or not according to the condition specified.

## Algorithm :-

Step 1 : Class starts

Step 2 : Define isLeapYear method

Step 3 : Define computeDate method

Step 4 : Main starts

Step 5 : Create a scanner object for input

Step 6 : Prompt and read dayNum, year, and n

Step 7 : Validate dayNum ( 1 to 366) and n ( 1 to 100); print error if out of range

Step 8 : Compute the initial date using computeDate

Step 9 : Add n days to dayNum; handle year rollover for leap/non-leap years.

Step 10 : Compute the new date after n days

Step 11 : Print the initial date and the new date after n days

Step 12 : Main ends

Step 13 : Class ends

# Variable Description :-

| Sl. No | Variable name | Datatype | Purpose |
|---|---|---|---|
| 1 | y | int | Determines if the given year is a leap year. |
| 2 | ret | boolean | Indicates if the year is a leap year (true/false) |
| 3 | day | int | Day number to be converted into a date. |
| 4 | year | int | Specifies the year for the date computation. |
| 5 | monthDays | int[] | Holds the number of days in each month. |
| 6 | monthNames | String[] | Stores the names of the months. |
| 7 | leap | boolean | Indicates if the year is a leap year. |
| 8 | i | int | Used to iterate through the months. |

| | | | |
|---|---|---|---|
| 9 | daySum | int | Helps in determining the month and day. |
| 10 | date | int | Stores the computed day of the month. |
| 11 | sb | StringBuffer | Builds the final date string. |
| 12 | in | Scanner | Reads input from the user. |
| 13 | dayNum | int | Stores the initial day number provided by the user. |
| 14 | n | int | Stores the number of days to add. |
| 15 | dateStr | String | Stores the initial date formatted as a string. |
| 16 | nDays | int | Stores the day number after adding n days. |
| 17 | nYear | int | Stores the adjusted year after adding n days. |

| | | | |
|---|---|---|---|
| 18 | nDateStr | String | Stores the new date formatted as a string. |

Q17) Write a program to input and output a natural number less than 1000 and output it in words.

## Algorithm :-

Step 1 : Class starts

Step 2 : Input number

Step 3 : Check if number n is in range (1 to 999). Print error if not.

Step 4 : Extract hundreds, tens and units digits then convert digits to words concatenating them and print the final string

Step 5 : Define a method to convert digits into words

Step 6 : Main starts

Step 7 : Create an object

Step 8 : Call inputNumber and extract methods

Step 9 : Main ends

Step 10 : Class ends

# Variable Description :-

| Sl No. | Variable Name | Datatype | Purpose |
|---|---|---|---|
| 1 | n | int | Stores the number to be converted to words |
| 2 | inp | BufferedReader | Reads input from the user |
| 3 | str | String | Holds the word for hundreds digit |
| 4 | str1 | String | Holds the word for the tens digit |
| 5 | str2 | String | Holds the word for the units digit or teen numbers |
| 6 | hund | int | Extract the hundreds digit |
| 7 | tens | int | Extract the tens digit |
| 8 | units | int | Extract the units digit |
| 9 | i | int | Iterates through each digit of the number |

| | | | |
|---|---|---|---|
| 10 | x | int | Used to determine the word for the digit |
| 11 | s | String | Holds the word equivalent of the digit or number |

Q18) Write a program to declare a matrix A[][] of order (mxn) where m is the number of rows and n is the number of columns such that both m and n must be greater than 2 and less than 20. Allow the user to input positive integers into this matrix. Perform the following tasks on the matrix :

a. Sort the elements of the outer row and column elements in ascending order using any standard sorting technique and arrange them in an array.

b. Calculate the sum of the outer row and column elements

c. Output the original matrix, rearranged matrix and only the boundary elements of the rearranged array with their sum.

Test your program with some random data.

# Algorithm :-

Step 1 : Class starts

Step 2 : Main starts

Step 3 : Input dimensions of the array

Step 5 : Initialize and input matrix

Step 6 : Print original matrix

Step 7 : Extract and sort outer elements

Step 8 : Calculate sum of outer elements

Step 9 : Rearrange matrix with sorted outer elements

Step 9 : Print the rearranged matrix, sorted outer matrix and sum of outer matrix

Step 10 : Main ends

Step 11 : Class ends

## Variable Description :-

| **Sl No.** | **Variable Name** | **Datatype** | **Purpose** |
|---|---|---|---|
| 1 | m | int | Stores the number of rows for the matrix |
| 2 | n | int | Stores the number of columns for the matrix |
| 3 | matrix | int[][] | Stores the elements of the matrix |
| 4 | outerElements | int[] | Stores the elements from the outer boundary of the matrix |
| 5 | sumOuter | int | Accumulates the sum of the outer elements |
| 6 | rearrangedMatrix | int[][] | Stores the rearranged matrix with sorted outer elements |
| 7 | index | int | Tracks the position while rearranging |

| | | | |
|---|---|---|---|
| | | | matrix elements |
| 8 | num | int | Used to iterate and sum the elements from the array |
| 9 | row | int[] | Used for printing each row of the matrix |
| 10 | i | int | Used to iterate through matrix rows |
| 11 | j | int | Used to iterate through matrix columns |

Q19) Numbers have different representation depending on the bases on which they are expressed. For example in base 3, the number 12 is written as 110 (1x3^2+1x3^1+0x3^0), but in base 8 it is written as 14(1x8^1+4x8^0).

Consider, for example, the integers 12 and 5. Certainly these are not equal if base 10 is used for each. But suppose 12 was a base 3 number and 5 was a base 6 number then what happens, 12 base 3= 1x3^1 + 2x3^0, or 5 base 6 or 5 base 10 (5 in any base is equal to 5 base 10). So 12 and 5 can be equal if you select the right bases for each of them.

Write a program to input two integers, x and y and calculate the smallest base for x and smallest base for y (likely different from x) so that x and y represent the same value. The base associated with x and y will be between 1 and 20 (both inclusive). In representing these numbers the digits 0 to 9 have their usual decimal interpretations. The uppercase alphabetic characters A through J represent digits 10 through 19 respectively.

# Algotithm :-

Step 1 : Class starts

Step 2 : Main starts

Step 3 : Input strings x and y

Step 4 : Find highest dihits in x and y (xMax and yMax)

Step 5 : Convert x to decimal for bases from xMax+1 to 20

Step 6 : Convert y to decimal for bases from yMax+1 to 20

Step 7 : Compare the stored decimal values if x

Step 8 : Print matching bases if found

Step 9 : Print “not equal in any base” message if no match

Step 10 : Main ends

Step 11 : Class ends

# Variable Description :-

| Sl. No. | Variable name | Datatype | Purpose |
|---|---|---|---|
| 1 | in | Scanner | Reads input from the user |
| 2 | x | String | Stores the first input number |
| 3 | y | String | Stores the second input number |
| 4 | xMax | int | Determines the minimum base for x |
| 5 | yMax | int | Determines the minimum base for y |
| 6 | found | boolean | Tracks if a matching base is found |
| 7 | decimalArr | int[] | Stores decimal conversions of x for comparison |
| 8 | i | int | Iterates through possible bases for y |
| 9 | j | int | Iterates through stored decimal values of x |

| | | | |
|---|---|---|---|
| 10 | t | int | Stores decimal conversion of y |
| 11 | numStr | String | Holds the number string for conversion |
| 12 | base | int | Specifies the base for conversion |
| 13 | num | int | Stores the converted decimal value |
| 14 | len | int | Determines the length of the number string |
| 15 | mul | int | Calculates positional value for conversion |
| 16 | ch | char | Represents individual digit character |
| 17 | d | int | Stores the numeric value of the character |
| 18 | c | char | Respesents character to convert to digit |
| 19 | value | int | Stores the converted digit value |

| | | | |
|---|---|---|---|
| 20 | numStr | String | Holds the number string to find highest digit |
| 21 | high | int | Tracks the highest digit value |

Q20) Write a program which takes a string (maximum 80 characters) terminated by a full stop. The words in this string are assumed to be separated by one or more blanks.

Arrange the words of the input string in descending order of their length. Same length words should be sorted alphabetically. Each word must start with an uppercase letter and the sentence should be terminated by a full stop.

## Algorithm :-

Step 1 : Class starts

Step 2 : Main starts

Step 3 : Read a string s from the user, appending a space at the end

Step 4 : Initialize temp and newS as empty strings

Step 5 : sort words by length

Step 6 : Print the sorted string newS

Step 7 : Main ends

Step 8 : Class ends

# Variable Description :-

| **Sl.No.** | **Variable name** | **Datatype** | **Purpose** |
|---|---|---|---|
| 1 | s | String | Stores the input string with an appended space |
| 2 | temp | String | Collects characters to form words |
| 3 | newS | String | Accumulates words in sorted order by length |
| 4 | i | int | Determines the current length of words to find |
| 5 | j | int | Iterates through each character in the input string |
| 6 | c | char | Represents individual character being processed |

Q21) A class Revstr defines a recursive function to reverse a string and check whether it is a Palindrome. The details of the class are given below :-

Class name : Revstr

Data members :-

str : stores the string

revst : stores the reverse of the string

Member functions :-

void getstr() : to accept the string

void recReverse(int) : to reverse the string using the recursive technique

void check() : to display the original string, its recursive and whether the string is a Palindrome or not

Specify the class Revstr giving the details of the functions void getStr(), void recReverse(int) and void check().

# Algorithm :-

Step 1 : Class starts

Step 2 : Initialize variables str, revst, and l to empty values and zero

Step 3 : Input word

Step 4 : Check for palindrome

Step 5 : Reverse the string

Step 6 : Main starts

Step 7 : Create an object of revstring

Step 8 : Call input to read the word

Step 9 : Call check to check if the word is a palindrome

Step 10 : Main ends

Step 11 : Class ends

# Variable Description :-

| **Sl. No.** | **Variable name** | **Datatype** | **Purpose** |
|---|---|---|---|
| 1 | ob | Scanner | Reads the input from the user |
| 2 | str | String | Stores the word to be checked for palindrome. |
| 3 | revst | String | Stores the reversed word. |
| 4 | l | int | Stores the length of the word. |
| 5 | n | int | Tracks the position in the string during recursion. |
| 6 | ch | char | Represents the character being processed during reversal. |

Q22) A financial institution has only two kinds of account Simple and Compound. The class has two protected attributes Acno (account) and p (principal) to store account number and money deposited initially. The interest of simple account of calculated by using SI = (P * R* T)/ 100 and for compound account is calculated by using CI = p (I + R/100)^T. R and T might vary for 2 kinds of account. Model the above situation by defining classes Account write appropriate constructors for passing attributes for these classes. Write the function double interest that calculates and returns interests for the classes Simple and Compound Also write the function display for all the three classes which prints all the attributes. Write main.

## Algorithm :-

Step 1 : Define base class Account

Step 2 : Initialize accountNumber and principle

Step 3 : Define constructor method to print account details

Step 4 : Define derived class Simple

Step 5 : Extend Account class

Step 6 : Initialize rate and time

Step 7 : Define constructor to set values

Step 8 : Define interest method to calculate simple interest

Step 9 : Override display method to include simple interest details

Step 10 : Define derived class Compound

Step 11 : Continue steps 5-9

Step 12 : Main starts

Step 13 : Create instances Simple and Compound accounts

Step 14 : Display details of both accounts

Step 15 : Main ends

Step 16 : Class ends

# Variable Description :-

| Sl.No | Variable name | Datatype | Purpose |
|---|---|---|---|
| 1 | accountNumber | String | Stores the account number |
| 2 | principle | double | Stores the principle account |
| 3 | rate | double | Stores the interest rate |
| 4 | time | double | Stores the time period |
| 5 | simpleAccount | Simple | Represents a simple interest account |
| 6 | compoundAccount | Compound | Represents a compound interest account |

Q23) A positive integer where the digits are neither in increasing nor decreasing order is called a bouncy number. In other words, if the digits of the number are unsorted, then it is bouncy. For example, 123742, 101, 43682, etc.

Design a program to accept a number (containing only digits 0 to 9) from the user. For an invalid input, display an appropriate message. Check whether the number is an increasing, decreasing or a bouncy number and display with an appropriate message in the format specified below:

Example 1

INPUT: Enter a number: 122344

OUTPUT: 122344 IS AN INCREASING NUMBER

Example 2

INPUT: Enter a number: 887552

OUTPUT: 887552 IS A DECREASING NUMBER

Example 3

INPUT: Enter a number: 185349

OUTPUT: 185349 IS A BOUNCY NUMBER

Example 4

INPUT: Enter a number: 98#57-649

OUTPUT: INVALID INPUT

# Algorithm :-

Step 1 : Class starts

Step 2 : Main starts

Step 3 : Input a number

Step 4 : Check if the number contains only digits. If not then print “INVALID INPUT” and return

Step 5 : Initialize ascending and descending flags as false. Loop through characters in the number and compare.

Step 6 : Set ascending to true if the current character is greater than the previous one and descending to true if the current character is less than the previous one.

Step 7 : If both ascending and descending are true, print bouncy number else of only ascending is true, print increasing number otherwise print decreasing number

Step 8 : Main ends

Step 9 : Class ends

# Variable Description :-

| Sl.No. | Variable name | Datatype | Purpose |
|---|---|---|---|
| 1 | in | Scanner | Reads input from the user |
| 2 | num | String | Stores the input number |
| 3 | ascending | boolean | Tracks if the number has ascending sequence |
| 4 | descending | boolean | Tracks if the number has descending sequence |
| 5 | i | int | Iterates through each character in num |
| 6 | previous | char | Represents the previous character during comparison |
| 7 | current | char | Represents the current character during comparison |
| 8 | s | String | Stores the string to be validated |
| 9 | ch | char | Represents the character being validated |

Q24) Write a program to accept a paragraph containing two sentences only. The sentences may be terminated by either '.', '?' or '!' only. Any other character may be ignored. The words are to be separated by a single blank space and are in uppercase.

Perform the following tasks:

(a) Accept the paragraph and check for validity.

(b) Obtain the length/size of the two sentences separately (number of words).

(c) Find the common words which occur in both the sentences.

Test your program with the sample data and some random data:

Example 1

INPUT: IS IT CLOUDY? IT MAY RAIN BECAUSE IT IS CLOUDY.

OUTPUT:

SENTENCE 1: 2 WORDS

SENTENCE 2: 7 WORDS

COMMON WORDS: IS IT CLOUDY

Example 2

INPUT: TO BE OR NOT TO BE. TO LET GO AND BE HAPPY.

OUTPUT:

SENTENCE 1: 6 WORDS

SENTENCE 2: 6 WORDS

COMMON WORDS: TO BE

Example 3

INPUT: IF YOU DRIVE FAST, YOU MAY MEET WITH AN ACCIDENT.

OUTPUT: INVALID PARAGRAPH

Example 4

INPUT: HOW ARE YOU@

OUTPUT: INVALID INPUT

# Algorithm :-

Step 1 : Class starts

Step 2 : Main starts

Step 3 : Read paragraph from the user, convert to uppercase and trim

Step 4 : Check if it ends with ‘.’, ‘?’, or ‘!’. Print invalid if not

Step 5 : Use StringTokenizer to split p into sentences using ".?!"

Step 6 : Check if there are exactly 2 sentences. Print invalid if not

Step 7 : Extract words and store in different arrays

Step 8 : Find common words

Step 9 : Print the number of words in each sentence and the common words

Step 10 : Main ends

Step 11 : Class ends

# Variable Description :-

| Sl.No. | Variable name | Datatype | Purpose |
|---|---|---|---|
| 1 | p | String | Stores the input paragraph |
| 2 | last | char | Checks the end of the paragraph for validity |
| 3 | st | StringTokenizer | Splits the paragraph into sentences |
| 4 | count | int | Counts the number of sentences |
| 5 | s1 | String | Stores the first sentence |
| 6 | s2 | String | Stores the second sentence |
| 7 | s | StringTokenizer | Splits sentences into words |
| 8 | w1 | String[] | Stores words of the first sentence |
| 9 | w2 | String[] | Stores words of the second sentence |
| 10 | common | String[] | Stores common words found in both sentences |
| 11 | index | int | Tracks the position in the common array |

| | | | |
|---|---|---|---|
| 12 | w | String | Represents the word being searched for |
| 13 | a | String[] | Represents the array of words being searched |

Q25) A Goldbach number is a positive even integer that can be expressed as the sum of two odd primes.

Note: All even integer numbers greater than 4 are Goldbach numbers.

Example:

6 = 3 + 3

10 = 3 + 7

10 = 5 + 5

Hence, 6 has one odd prime pair 3 and 3. Similarly, 10 has two odd prime pairs, i.e. 3 and 7, 5 and 5.

Write a program to accept an even integer ‘N’ where N > 9 and N < 50. Find all the odd prime pairs whose sum is equal to the number ‘N’.

Test your program with the following data and some random data:

Example 1:

INPUT: N = 14

OUTPUT:

PRIME PAIRS ARE:

3, 11

7, 7

Example 2:

INPUT: N = 30

OUTPUT:

PRIME PAIRS ARE:

7, 23

11, 19

13, 17

Example 3:

INPUT: N = 17

OUTPUT:

INVALID INPUT. NUMBER IS ODD.

Example 4:

INPUT: N = 126

OUTPUT:

INVALID INPUT. NUMBER OUT OF RANGE.

# Algorithm :-

Step 1 : Class starts

Step 2 : Main starts

Step 3 : Input number

Step 4 : Check if the number is greater than 50 or not even. Print invalid input if so

Step 5 : Find prime pairs

Step 6 : Check if the number is prime by counting its didvisors

Step 7 : Main ends

Step 8 : Class ends

## Variable Description :-

| Sl.No. | Variable name | Datatype | Purpose |
|---|---|---|---|
| 1 | n | int | Stores the input number |
| 2 | i | int | Iterates through possible prime pairs |
| 3 | f | int | Tracks the number of divisors |