This is to certify that \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is a bona fide student of XII Science A. He has successfully completed the computer science project in JAVA language during the session 2024 **-** 25 for the fulfillment of ISC exam by the council.

**\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_**

**Visiting Examiner Internal Examiner**

**\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_**

**Date Date**

**Question 1:**

A **Prime-Adam** integer is a positive integer (without leading zeros) which is a prime as well as an Adam number.

**Prime number**: A number which has only two factors, i.e. 1 and the number itself. Example: 2, 3, 5, 7 ... etc.

**Adam number**: The square of a number and the square of its reverse are reverse to each other.

Example: If n = 13 and reverse of 'n' = 31, then,

(13)2 = 169

(31)2 = 961 which is reverse of 169

thus 13, is an Adam number.

Accept two positive integers m and n, where m is less than n as user input. Display all Prime-Adam integers that are in the range between m and n (both inclusive) and output them along with the frequency, in the format given below:

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

**Example 1**

**INPUT:**  
m = 5  
n = 100

**OUTPUT:**  
THE PRIME-ADAM INTEGERS ARE:  
11 13 31  
FREQUENCY OF PRIME-ADAM INTEGERS IS: 3

**Example 2**

**INPUT:**  
m = 100  
n = 200

**OUTPUT:**  
THE PRIME-ADAM INTEGERS ARE:  
101 103 113  
FREQUENCY OF PRIME-ADAM INTEGERS IS: 3

**Example 3**

**INPUT:**  
m = 50  
n = 70

**OUTPUT:**  
THE PRIME-ADAM INTEGERS ARE:  
NIL  
FREQUENCY OF PRIME-ADAM INTEGERS IS: 0

**Example 4**

**INPUT:**  
m = 700  
n = 450

**OUTPUT:**  
INVALID INPUT

**Algorithm:**

1. **Input Validation**:
   * Accept the inputs m and n. If m is greater than n, display "INVALID INPUT" and exit the program.
2. **Prime Check**:
   * For every number i in the range from m to n (both inclusive), check if i is a prime number.
   * A number is prime if it has only two factors: 1 and itself. You can check divisibility from 2 to the square root of i.
3. **Reverse and Adam Check**:
   * If i is prime, reverse the number i.
   * Square both i and its reverse.
   * If the square of the reverse of i is the reverse of the square of i, then the number is an Adam number.
4. **Print Prime-Adam Integers**:
   * Print the found Prime-Adam numbers as the loop progresses.
   * Count the frequency of such numbers when one is found.
5. **Print NIL case and frequency**:
   * If there are no Prime-Adam integers (count = 0), output "NIL" and frequency 0.
   * Else output the frequency (count).

**Solution:**

import java.util.Scanner;

public class PrimeAdam {

*// Function to reverse a number*

public static int reverse(int *num*) {

int rev = 0;

while (num > 0) {

rev = rev \* 10 + num % 10;

num /= 10;

}

return rev;

}

*// Function to check if a number is prime*

public static boolean isPrime(int *num*) {

if (num < 2) return false;

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) return false;

}

return true;

}

*// Function to check if a number is an Adam number*

public static boolean isAdam(int *num*) {

int rev = reverse(num);

return num \* num == reverse(rev \* rev);

}

public static void main(String[] *args*) {

Scanner sc = new Scanner(System.in);

*// Input m and n*

System.out.print("Enter m: ");

int m = sc.nextInt();

System.out.print("Enter n: ");

int n = sc.nextInt();

sc.close();

*// Check for invalid input*

if (m > n) {

System.out.println("INVALID INPUT");

return;

}

System.out.println("THE PRIME-ADAM INTEGERS ARE:");

int count = 0;

*// Loop through the range [m, n]*

for (int i = m; i <= n; i++) {

*// Check if number is prime and Adam and print it*

if (isPrime(i) && isAdam(i)) {

count++;

System.out.print(i + " ");

}

}

*// Output the results*

if (count == 0) {

System.out.println("NIL");

}

System.out.println("\nFREQUENCY OF PRIME-ADAM INTEGERS IS: " + count);

}

}

**Output:**

**Variable Description Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Scope** | **Name** | **Data Type** | **Description** |
| main | **m** | **int** | Starting range value input by the user |
| main | **n** | **int** | Ending range value input by the user |
| main | **i** | **int** | Loop variable to iterate from m to n |
| reverse | **rev** | **int** | Stores the reversed value of a number |
| main | **count** | **int** | Counts the number of Prime-Adam integers found |

**Question 2:**

Write a program to declare a matrix A[ ][ ] of order (M x N) where 'M' is the number of rows and 'N' is the number of columns such that the value of 'M' must be greater than 0 and less than 10 and the value of 'N' must be greater than 2 and less than 6. Allow the user to input digits (0 - 7) only at each location, such that each row represents an octal number.

Example:

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 3 | 1 | (decimal equivalent of 1st row = 153 i.e. 2x82 + 3x81 + 1x80) |
| 4 | 0 | 5 | (decimal equivalent of 2nd row = 261 i.e. 4x82 + 0x81 + 5x80) |
| 1 | 5 | 6 | (decimal equivalent of 3rd row = 110 i.e. 1x82 + 5x81 + 6x80) |

Perform the following tasks on the matrix:

1. Display the original matrix.
2. Calculate the decimal equivalent for each row and display as per the format given below.

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

**Example 1:**

**INPUT:**  
M = 1  
N = 3  
ENTER ELEMENTS FOR ROW 1: 1 4 4

**OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| **FILLED MATRIX** | | | **DECIMAL EQUIVALENT** |
| 1 | 4 | 4 | 100 |

**Example 2:**

**INPUT:**  
M = 3  
N = 4  
ENTER ELEMENTS FOR ROW 1: 1 1 3 7  
ENTER ELEMENTS FOR ROW 2: 2 1 0 6  
ENTER ELEMENTS FOR ROW 3: 0 2 4 5

**OUTPUT:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FILLED MATRIX** | | | | **DECIMAL EQUIVALENT** |
| 1 | 1 | 3 | 7 | 607 |
| 2 | 1 | 0 | 6 | 1094 |
| 0 | 2 | 4 | 5 | 165 |

**Example 3:**

**INPUT:**  
M = 3  
N = 3  
ENTER ELEMENTS FOR ROW 1: 2 4 8

**OUTPUT:**  
INVALID INPUT

**Example 4:**

**INPUT:**  
M = 4  
N = 6

**OUTPUT:**  
OUT OF RANGE

**Algorithm:**

1. **Input Validation**:
   * Accept the inputs M and N. Ensure that M is between 1 and 9, and N is between 3 and 5. If the input is out of range, display "OUT OF RANGE" and terminate the program.
2. **Matrix Declaration and Input**:
   * Declare a matrix A[M][N] of size M by N.
   * For each element in the matrix, prompt the user to input a digit between 0 and 7. If any element is outside this range, display "INVALID INPUT" and terminate the program.
3. **Display the Original Matrix**:
   * Display the matrix in a tabular form as provided by the user.
4. **Convert Each Row to Decimal**:
   * For each row in the matrix, convert it from an octal number to its decimal equivalent. This can be done using the formula:
   * Calculate the decimal equivalent for each row.
5. **Display the Decimal Equivalents**:
   * Print the original matrix along with its decimal equivalent for each row.

**Solution:**

import java.util.Scanner;

public class OctalMatrix {

public static void main(String[] *args*) {

Scanner sc = new Scanner(System.in);

*// Input M and N values*

System.out.print("Enter number of rows M: ");

int M = sc.nextInt();

System.out.print("Enter number of columns N: ");

int N = sc.nextInt();

sc.close();

*// Check for valid range of M and N*

if (M <= 0 || M >= 10 || N <= 2 || N >= 6) {

System.out.println("OUT OF RANGE");

return;

}

*// Declare the matrix*

int[][] A = new int[M][N];

*// Input elements for the matrix and validate each element*

for (int i = 0; i < M; i++) {

System.out.println("Enter elements for row" +(i+1)+ ": ");

for (int j = 0; j < N; j++) {

A[i][j] = sc.nextInt();

*// Check if input is a valid octal digit (0-7)*

if (A[i][j] < 0 || A[i][j] > 7) {

System.out.println("INVALID INPUT");

return;

}

}

}

*// Display the filled matrix and the decimal equivalent*

System.out.println("FILLED MATRIX\tDECIMAL EQUIVALENT");

for (int i = 0; i < M; i++) {

int decimal\_value = 0;

*// Display the row and calculate its decimal equivalent*

for (int j = 0; j < N; j++) {

System.out.print(A[i][j] + "\t");

decimal\_value += A[i][j] \* Math.pow(8, N - j - 1);

}

*// Print the decimal equivalent for the row*

System.out.println(decimal\_value);

}

}

}

**Variable Description Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| **M** | **int** | Number of rows, input by the user |
| **N** | **int** | Number of columns, input by the user |
| **A** | **int[][]** | Matrix to store the digits |
| **i** | **int** | Loop variable for iterating through rows |
| **j** | **int** | Loop variable for iterating through columns |
| **decimal\_value** | **int** | Stores the decimal equivalent of a row |
| **octal\_digit** | **int** | Octal digit input by the user at each matrix location |

**Question 3:**

Write a program to accept a sentence which may be terminated by either '.', '?' or '!' only. The words are to be separated by a single blank space and are in UPPER CASE.

Perform the following tasks:

1. Check for the validity of the accepted sentence only for the terminating character.
2. Arrange the words in ascending order of their length. If two or more words have the same length, then sort them alphabetically.
3. Display the original sentence along with the converted sentence.

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

**Example 1:**

**INPUT:**  
AS YOU SOW SO SHALL YOU REAP.

**OUTPUT:**  
AS YOU SOW SO SHALL YOU REAP.  
AS SO SOW YOU YOU REAP SHALL

**Example 2:**

**INPUT:**  
SELF HELP IS THE BEST HELP.

**OUTPUT:**  
SELF HELP IS THE BEST HELP.  
IS THE BEST HELP HELP SELF

**Example 3:**

**INPUT:**  
BE KIND TO OTHERS.

**OUTPUT:**  
BE KIND TO OTHERS.  
BE TO KIND OTHERS

**Example 4:**

**INPUT:**  
NOTHING IS IMPOSSIBLE#

**OUTPUT:**  
INVALID INPUT

**Algorithm:**

1. **Input the Sentence**:
   * Read a sentence from the user using the Scanner class.
   * The sentence must be in uppercase and can only end with '.', '?', or '!' to be considered valid.
2. **Validate the Sentence**:
   * Use the isValidString() function to check the following conditions:
     + The sentence should not be null or empty.
     + The last character must be either '.', '?', or '!'.
   * If the sentence does not meet these conditions, print "INVALID INPUT" and terminate the program.
3. **Print the Original Sentence**:
   * If the sentence is valid, print the original sentence.
4. **Remove the Terminating Character**:
   * Remove the last character ('.', '?', or '!') from the input sentence for further processing.
5. **Split the Sentence into Words**:
   * Split the sentence into words using space (" ") as the delimiter.
6. **Sort the Words**:
   * Sort the words by their length in ascending order.
   * If two words have the same length, sort them alphabetically.
   * The sortString() function handles the sorting using a modified bubble sort approach and the swap() function to swap words.
7. **Concatenate the Sorted Words**:
   * After sorting, concatenate the words back into a single sentence with a space between each word.
8. **Print the Sorted Sentence**:
   * Display the sentence with words arranged based on the sorting criteria.

**Solution:**

import java.util.Scanner;

public class StringCheck {

*// Method to check if the sentence ends with '.', '?', or '!'*

public static boolean isValidString(String str) {

int len = str.length();

char last\_ch = str.charAt(len - 1);

if (str == null || len == 0 ||

(last\_ch != '.' && last\_ch != '?' && last\_ch != '!')) {

return false;

}

return true;

}

*// Method to swap two elements in an array*

public static void swap(String[] words, int i, int j) {

String temp = words[i];

words[i] = words[j];

words[j] = temp;

}

*// Method to sort words by length, then alphabetically*

public static String sortString(String ipStr) {

String words[] = ipStr.split(" ");

int wordCount = words.length;

for (int i = 0; i < wordCount - 1; i++) {

for (int j = i + 1; j < wordCount; j++) {

if (words[i].length() > words[j].length() ||

(words[i].length() == words[j].length() &&

words[i].compareTo(words[j]) > 0)) {

swap(words, i, j);

}

}

}

String sorted = ""; *// string to be returned*

for (int i = 0; i < wordCount - 1; i++) {

sorted += words[i] + " ";

}

*// Append last word without space*

sorted += words[wordCount - 1];

return sorted;

}

public static void main(String args[]) {

Scanner in = new Scanner(System.in);

System.out.println("Enter a sentence:");

String str = in.nextLine();

int len = str.length();

System.out.println();

in.close();

if (!isValidString(str)) {

System.out.println("INVALID INPUT");

return;

}

System.out.println(str);

*// Remove the terminating character ('.', '?', '!')*

str = str.substring(0, len - 1);

String sortedStr = sortString(str);

System.out.println(sortedStr);

}

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| **str** | **String** | Stores the sentence input by the user. |
| **words** | **String[]** | Stores the array of words split from the sentence. |
| **wordCount** | **int** | Holds the number of words in the sentence. |
| **last\_ch** | **char** | Stores the last character of the input sentence for validation. |
| **len** | **int** | Holds the length of the input sentence. |
| **sorted** | **String** | Stores the concatenated string of sorted words. |
| **temp** | **String** | Temporary variable for swapping words during sorting. |
| **i, j** | **int** | Loop variables used for sorting and word processing. |

**Question 1:**

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 date.

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.

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

**Example 1**

**INPUT:**  
DAY NUMBER: 255  
YEAR: 2018  
DATE AFTER (N DAYS): 22

**OUTPUT:**  
DATE: 12TH SEPTEMBER, 2018  
DATE AFTER 22 DAYS: 4TH OCTOBER, 2018

**Example 2**

**INPUT:**  
DAY NUMBER: 360  
YEAR: 2018  
DATE AFTER (N DAYS): 45

**OUTPUT:**  
DATE: 26TH DECEMBER, 2018  
DATE AFTER 45 DAYS: 9TH FEBRUARY, 2019

**Example 3**

**INPUT:**  
DAY NUMBER: 500  
YEAR: 2018  
DATE AFTER (N DAYS): 33

**OUTPUT:**  
DAY NUMBER OUT OF RANGE

**Example 4**

**INPUT:**  
DAY NUMBER: 150  
YEAR: 2018  
DATE AFTER (N DAYS): 330

**OUTPUT:**  
DATE AFTER (N DAYS) OUT OF RANGE

**Algorithm:**

1. **Input Validation**:
   * Accept dayNumber, year, and N from the user.
   * Check if dayNumber is between 1 and 366 for a leap year, and 1 and 365 for a non-leap year.
   * Check if N is between 1 and 100.
   * If the input is invalid, display an appropriate error message and terminate.
2. **Leap Year Calculation**:
   * A year is a leap year if it is divisible by 4 but not divisible by 100 unless divisible by 400.
3. **Date Calculation**:
   * Create an array of days in each month for both leap years and non-leap years.
   * Using the dayNumber, calculate the corresponding month and day in that month.
4. **Future Date Calculation**:
   * Add N days to the initial date.
   * Adjust the month and year if the number of days exceeds the number of days in the current month.
   * If the month exceeds 12, increment the year accordingly.
5. **Edge Cases**:
   * Ensure that if the date after N days exceeds the given year’s day limit or dayNumber is out of range, appropriate error messages are displayed.

**Solution:**

import java.util.Scanner;

public class DateCalculator {

*// Checks if a year is a leap year.*

  public static boolean isLeapYear(int *y*) {

    return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);

  }

*// Computes the date given a day number and year.*

  public static String computeDate(int *day*, int *year*) {

    int[] monthDays = { 31, 28, 31, 30, 31,

30, 31, 31, 30, 31, 30, 31 };

    String[] monthNames = {

      "JANUARY", "FEBRUARY", "MARCH", "APRIL",

      "MAY", "JUNE", "JULY", "AUGUST",

      "SEPTEMBER", "OCTOBER", "NOVEMBER", "DECEMBER"

    };

*// Adjust for February in leap year*

    monthDays[1] += isLeapYear(year) ? 1 : 0;

    int i = 0, daySum = 0;

    for (i = 0; i < monthDays.length; i++) {

      daySum += monthDays[i];

      if (daySum >= day) break;

    }

    int date = day + monthDays[i] - daySum;

    return date + getDaySuffix(date) + " " + monthNames[i] +   
 ", " + year;

  }

*// Gets the suffix for a day number (ST, ND, RD, TH).*

  public static String getDaySuffix(int *day*) {

    if (day >= 11 && day <= 13) {

return "TH";

}

    switch (day % 10) {

      case 1:

return "ST";

      case 2:

return "ND";

      case 3:

return "RD";

      default:

return "TH";

    }

  }

  public static void main(String[] *args*) {

    Scanner in = new Scanner(System.in);

    System.out.print("DAY NUMBER: ");

    int dayNum = in.nextInt();

    System.out.print("YEAR: ");

    int year = in.nextInt();

    System.out.print("DATE AFTER (N DAYS): ");

    int n = in.nextInt();

    in.close();

    if (dayNum < 1 || dayNum > 366) {

      System.out.println("DAY NUMBER OUT OF RANGE");

      return;

    }

    if (n < 1 || n > 100) {

      System.out.println("DATE AFTER (N DAYS) OUT OF RANGE");

      return;

    }

*// Original date calculation*

    String dateStr = computeDate(dayNum, year);

*// Future date calculation*

    int nDays = dayNum + n;

    int nYear = year;

    boolean leap = isLeapYear(year);

    if (leap && nDays > 366) {

      nYear = nYear + 1;

      nDays = nDays - 366;

    } else if (!leap && nDays > 365) {

      nYear = nYear + 1;

      nDays = nDays - 365;

    }

    String nDateStr = computeDate(nDays, nYear);

*// Output results*

    System.out.println();

    System.out.println("DATE: " + dateStr);

    System.out.println("DATE AFTER " + n + " DAYS: " +   
 nDateStr);

  }

}

**Variable Description Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| **dayNum** | **int** | The input day number (1-366 for leap years, 1-365 for others) |
| **year** | **int** | The input year (4 digits) |
| **N** | **int** | Number of days to add to the original date |
| **monthDays** | **int[]** | Array holding the days in each month |
| **monthNames** | **String[]** | Array holding the names of each month |
| **leap** | **boolean** | Indicates if the input year is a leap year |
| **daySum** | **int** | Running sum of days to determine the month |
| **date** | **int** | The final day of the month after calculating |
| **i** | **int** | Loop variable to iterate through the monthDays array |
| **nDays** | **int** | The day number after adding N days |
| **nYear** | **int** | The year for the future date |
| **dateStr** | **String** | Formatted date for the initial day number |
| **nDateStr** | **String** | Formatted date after N days |

**Question 2:**

Write a program to declare a single-dimensional array a[ ] and a square matrix b[ ][ ] of size N, where N > 2 and N < 10. Allow the user to input positive integers into the single dimensional array.

Perform the following tasks on the matrix:

1. Sort the elements of the single-dimensional array in ascending order using any standard sorting technique and display the sorted elements.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 5 | 8 |
| 1 | 2 | 5 | 1 |
| 1 | 2 | 1 | 2 |
| 1 | 1 | 2 | 5 |

1. Fill the square matrix b[ ][ ] in the following format:  
   If the array a[ ] = {5, 2, 8, 1}   
   then, after sorting a[ ] = {1, 2, 5, 8}  
   Then, the matrix b[ ][ ] would fill as below:
2. Display the filled matrix in the above format.

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

**Example 1**

**INPUT:**  
N = 3  
ENTER ELEMENTS OF SINGLE DIMENSIONAL ARRAY: 3 1 7

**OUTPUT:**  
SORTED ARRAY: 1 3 7  
FILLED MATRIX

|  |  |  |
| --- | --- | --- |
| **1** | **3** | **7** |
| **1** | **3** | **1** |
| **1** | **1** | **3** |

**Example 2**

**INPUT:**  
N = 13

**OUTPUT:**  
MATRIX SIZE OUT OF RANGE

**Example 3**

**INPUT:**  
N = 5  
ENTER ELEMENTS OF SINGLE DIMENSIONAL ARRAY: 10 2 5 23 6

**OUTPUT:**  
SORTED ARRAY: 2 5 6 10 23  
FILLED MATRIX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 6 | 10 | 23 |
| 2 | 5 | 6 | 10 | 2 |
| 2 | 5 | 6 | 2 | 5 |
| 2 | 5 | 2 | 5 | 6 |
| 2 | 2 | 5 | 6 | 10 |

**Algorithm:**

1. **Input and Validation:**
   * Accept the size N for the matrix and array. Ensure N is between 3 and 9 (both inclusive). If out of range, display an error message and exit.
2. **Array Input:**
   * Accept N positive integers into the single-dimensional array a[ ].
3. **Sorting:**
   * Sort the array a[ ] in ascending order using the bubble sort algorithm.
4. **Matrix Filling:**
   * Fill the matrix b[ ][ ] such that:
     + First row b[0] holds the sorted array a[ ].
     + Each subsequent row has the same elements as the previous row but shifted cyclically leftward.
5. **Display:**
   * Display the sorted array.
   * Display the matrix.

**Solution:**

import java.util.Scanner;

public class Array {

*// Sorting array using bubble sort*

  public static void sortArray(int *arr*[]) {

    int n = arr.length;

    for (int i = 0; i < n - 1; i++) {

      for (int j = 0; j < n - i - 1; j++) {

        if (arr[j] > arr[j + 1]) {

          int t = arr[j];

          arr[j] = arr[j + 1];

          arr[j + 1] = t;

        }

      }

    }

  }

*// Function to fill matrix b[][] using cyclic shifts*

  public static void fillMatrix(int *a*[], int *b*[][]) {

    int n = a.length;

*// First row is the sorted array itself*

    for (int j = 0; j < n; j++) {

      b[0][j] = a[j];

    }

*// For each row cyclically shift prev row left by 1 position*

    for (int i = 1; i < n; i++) {

      for (int j = 0; j < n; j++) {

*// Shift elements cyclically*

        b[i][j] = b[i - 1][(j + 1) % n];

      }

    }

  }

public static void main(String *args*[]) {

    Scanner in = new Scanner(System.in);

    System.out.print("ENTER VALUE OF N: ");

    int n = in.nextInt();

    in.close();

    if (n <= 2 || n >= 10) {

      System.out.println("MATRIX SIZE OUT OF RANGE");

      return;

    }

    int a[] = new int[n], b[][] = new int[n][n];

    System.out.println("ENTER ELEMENTS OF SINGLE DIMENSIONAL ARRAY:");

    for (int i = 0; i < n; i++) {

      a[i] = in.nextInt();

    }

    sortArray(a);

    System.out.println("SORTED ARRAY:");

    for (int i = 0; i < n; i++) {

      System.out.print(a[i] + " ");

    }

    fillMatrix(a, b);

    System.out.println("\nFILLED MATRIX:");

    for (int i = 0; i < n; i++) {

      for (int j = 0; j < n; j++) {

        System.out.print(b[i][j] + " ");

      }

      System.out.println();

    }

  }

}

**Variable Description Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Scope** | **Name** | **Data Type** | **Description** |
| Global | **a[]** | **int[]** | Single-dimensional array to hold N positive integers. |
| Global | **b[][]** | **int[][]** | 2D square matrix of size N x N filled with sorted elements. |
| Local (main) | **n** | **int** | Size of the matrix (between 3 and 9). |
| Local (sortArray) | **i, j** | **int** | Loop variables used for bubble sorting. |
| Local (main) | **r, c** | **int** | Loop variables for filling and displaying the matrix. |
| Local (main) | **temp** | **int** | Temporary variable used for shifting rows cyclically. |

**Question 3**

Write a program to accept a sentence which may be terminated by either ‘.’, ‘?’ or ‘!’ only. The words are to be separated by a single blank space and are in uppercase.

Perform the following tasks:

(a) Check for the validity of the accepted sentence.

(b) Convert the non-palindrome words of the sentence into palindrome words by concatenating the word by its reverse (excluding the last character).

Example:

The reverse of the word HELP would be LEH (omitting the last alphabet) and by concatenating both, the new palindrome word is HELPLEH. Thus, the word HELP becomes HELPLEH.

Note: The words which end with repeated alphabets, for example ABB would become ABBA and not ABBBA and XAZZZ becomes XAZZZAX.

Palindrome word: Spells same from either side. Example: DAD, MADAM etc.

(c) Display the original sentence along with the converted sentence.

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

**Example 1**

**INPUT:**  
THE BIRD IS FLYING.

**OUTPUT:**  
THE BIRD IS FLYING.  
THEHT BIRDRIB ISI FLYINGNIYLF

**Example 2**

**INPUT:**  
IS THE WATER LEVEL RISING?

**OUTPUT:**  
IS THE WATER LEVEL RISING?  
ISI THEHT WATERETAW LEVEL RISINGNISIR

**Example 3**

**INPUT:**  
THIS MOBILE APP LOOKS FINE.

**OUTPUT:**  
THIS MOBILE APP LOOKS FINE.  
THISIHT MOBILELIBOM APPA LOOKSKOOL FINENIF

**Example 3**

**INPUT:**  
YOU MUST BE CRAZY#

**OUTPUT:**  
INVALID INPUT

**Algorithm:**

1. **Input Validation**:
   * Accept a sentence from the user.
   * Convert the sentence to uppercase.
   * Check if the sentence ends with a valid punctuation mark (i.e., ., ?, or !). If not, display "INVALID INPUT" and exit.
2. **Process Each Word**:
   * Remove the terminating punctuation.
   * Split the sentence into individual words using space as the delimiter.
3. **Palindrome Check**:
   * For each word, check if it's a palindrome:
     + If a word reads the same forwards and backwards, it’s a palindrome.
   * If the word is not a palindrome, convert it into one by reversing it (excluding the last character) and concatenating it to the original word. Handle words that end with repeated characters specially (i.e., avoid repeating the same character multiple times).
4. **Concatenate the Converted Words**:
   * Construct a new sentence by joining the converted palindrome words.
5. **Display**:
   * Display the original sentence and the transformed sentence.

**Solution:**

import java.util.Scanner;

public class Palindrome {

*// Function to check if a word is a palindrome*

  public static boolean isPalindrome(String *word*) {

    int len = word.length();

    for (int i = 0; i < len / 2; i++) {

      if (word.charAt(i) != word.charAt(len - 1 - i)) {

        return false;

      }

    }

    return true;

  }

*// Function to make a word palindrome*

*// character*

  public static String makePalindrome(String *word*) {

    int len = word.length(), i = len - 1;;

    char lastChar = word.charAt(len - 1);

*// Find the first character from the end that is not equal to the last character*

    while (i > 0 && word.charAt(i) == lastChar) {i--;}

*// Appending the reverse (up to character i)*

    String result = word;

    for (int j = i; j >= 0; j--) {

      result += word.charAt(j);

    }

    return result;

  }

  public static void main(String[] *args*) {

    Scanner in = new Scanner(System.in);

    System.out.println("ENTER THE SENTENCE:");

    String ipStr = in.nextLine().trim().toUpperCase();

    int len = ipStr.length();

    in.close();

*// Check if the input sentence ends with '.', '?', or '!'*

    char lastChar = ipStr.charAt(len - 1);

    if (lastChar != '.' && lastChar != '?' && lastChar != '!') {

      System.out.println("INVALID INPUT");

      return;

    }

*// Remove the punctuation at the end*

    String str = ipStr.substring(0, len - 1);

*// Split the sentence into words based on spaces*

    String[] words = str.split(" ");

    String result = "";

    for (int i = 0; i < words.length; i++) {

      String word = words[i];

*// If the word is already a palindrome, keep it as is*

      result += (isPalindrome(word) ? word : makePalindrome(word)) + " ";

    }

*// Output the original sentence with punctuation then result*

    System.out.println("\n" + ipStr);

    System.out.println(result);

  }

}

**Variable Description Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| **ipStr** | **String** | Input sentence entered by the user, trimmed and converted to uppercase. |
| **len** | **int** | Length of the input sentence. |
| **lastChar** | **char** | Last character of the input sentence ('.', '!', or '?'). |
| **str** | **String** | Sentence excluding the last character (punctuation). |
| **words[]** | **String[]** | Array of words obtained by splitting str based on spaces. |
| **isPalinWord** | **boolean** | Indicates if the current word is a palindrome. |
| **palinWord** | **String** | Palindrome version of the current word. |
| **i** | **int** | Loop index for iterating over the words and characters. |
| **result** | **String** | Sentence after converting non-palindrome words into palindrome words. |

**Question 1**

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:**

1. **Input Validation**:
   * Accept an integer input N.
   * Check if N is within the valid range (i.e., greater than 9 and less than 50).
   * If N is outside the range, print "INVALID INPUT. NUMBER OUT OF RANGE" and terminate.
   * If N is odd, print "INVALID INPUT. NUMBER IS ODD" and terminate.
2. **Prime Pair Search**:
   * Start with the smallest odd prime (a = 3).
   * For each odd prime a less than or equal to N/2, compute b = N - a.
   * Check if both a and b are prime.
   * If they are, print the pair a, b.
3. **Prime Checking**:
   * Implement a function isPrime() to check if a given number is prime. A number is prime if it has exactly two divisors: 1 and itself.
4. **Looping for Odd Primes**:
   * Increment a by 2 in each iteration to check only odd numbers.
   * Continue the loop until a exceeds N/2.
5. **Output**:
   * Display all valid prime pairs whose sum equals N.

**Solution:**

import java.util.Scanner;

public class GoldbachNumber {

*// Function to check if a number is prime*

  public static boolean isPrime(int *num*) {

*// Prime numbers are greater than 1*

if (num < 2) {

      return false;

}

    for (int i = 2; i <= num / 2; i++) {

*// If divisible by any num other than 1 and itself*

      if (num % i == 0) {

        return false;

      }

    }

*// Prime if not divisible by any num other than 1 and itself*

    return true;

  }

  public static void main(String[] *args*) {

    Scanner in = new Scanner(System.in);

*// Input the value of N*

    System.out.print("ENTER THE VALUE OF N: ");

    int n = in.nextInt();

    in.close();

*// Validate if N is within the range and even*

if (n <= 9 || n >= 50) {

      System.out.println("INVALID INPUT. NUMBER OUT OF RANGE.");

      return;

    }

    if (n % 2 != 0) {

      System.out.println("INVALID INPUT. NUMBER IS ODD.");

      return;

    }

*// Output prime pairs*

    System.out.println("PRIME PAIRS ARE:");

*// Start checking odd primes starting from 3*

    for (int a = 3; a <= n / 2; a += 2) {

*// Calculate b as the complement of a*

      int b = n - a;

      if (isPrime(a) && isPrime(b)) {

*// Print the prime pair*

        System.out.println(a + ", " + b);

      }

    }

  }

}

**Variable Description Table:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| **n** | **int** | The input number entered by the user. |
| **a** | **int** | First odd prime candidate in the pair. Starts at 3. |
| **b** | **int** | Second odd prime in the pair, calculated as b = n - a. |
| **c** | **int** | Counter used in isPrime function to count divisors of a number. |

**Question 2**

Write a program to declare a matrix a[ ][ ] of order (m × n) where 'm' is the number of rows and 'n' is the number of columns such that the values of both 'm' and 'n' must be greater than 2 and less than 10. Allow the user to input integers into this matrix.

Perform the following tasks on the matrix:

1. Display the original matrix.
2. Sort each row of the matrix in ascending order using any standard sorting technique.
3. Display the changed matrix after sorting each row.

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

**Example 1**

**INPUT:**  
M = 4  
N = 3

ENTER ELEMENTS OF MATRIX:

|  |  |  |
| --- | --- | --- |
| 11 | -2 | 3 |
| 5 | 16 | 7 |
| 9 | 0 | 4 |
| 3 | 1 | 8 |

**OUTPUT:**  
ORIGINAL MATRIX

|  |  |  |
| --- | --- | --- |
| 11 | -2 | 3 |
| 5 | 16 | 7 |
| 9 | 0 | 4 |
| 3 | 1 | 8 |

MATRIX AFTER SORTING ROWS

|  |  |  |
| --- | --- | --- |
| -2 | 3 | 11 |
| 5 | 7 | 16 |
| 0 | 4 | 9 |
| 1 | 3 | 8 |

**Example 2**

**INPUT:**  
M = 3  
N = 3

ENTER ELEMENTS OF MATRIX

|  |  |  |
| --- | --- | --- |
| 22 | 5 | 19 |
| 7 | 36 | 12 |
| 9 | 13 | 6 |

**OUTPUT:**  
ORIGINAL MATRIX

|  |  |  |
| --- | --- | --- |
| 22 | 5 | 19 |
| 7 | 36 | 12 |
| 9 | 13 | 6 |

MATRIX AFTER SORTING ROWS

|  |  |  |
| --- | --- | --- |
| 5 | 19 | 22 |
| 7 | 12 | 36 |
| 6 | 9 | 13 |

**Example 3**

**INPUT:**  
M = 11  
N = 5

**OUTPUT:**  
MATRIX SIZE OUT OF RANGE.

**Algorithm:**

1. **Input Matrix Dimensions:**
   * Read values for m (number of rows) and n (number of columns).
   * Validate that m and n are both greater than 2 and less than 10. If not, display "MATRIX SIZE OUT OF RANGE" and terminate.
2. **Input Matrix Elements:**
   * Create a 2D array a[m][n] to store the matrix elements.
   * Use a nested loop to allow the user to input values for each element in the matrix.
3. **Display Original Matrix:**
   * Traverse through the matrix and print its original form, row by row.
4. **Sort Each Row:**
   * Implement bubble sort on each row of the matrix. For each row, compare adjacent elements and swap them if they are out of order. Repeat this process until all elements in the row are sorted.
5. **Display Sorted Matrix:**
   * After sorting, traverse the matrix again and print each row.

**Solution:**

import java.util.Scanner;

public class ArraySort {

  public static void main(String *args*[]) {

    Scanner in = new Scanner(System.in);

*// Input number of rows (m) and columns (n)*

    System.out.print("ENTER THE VALUE OF M: ");

    int m = in.nextInt();

    System.out.print("ENTER THE VALUE OF N: ");

    int n = in.nextInt();

    in.close();

*// Check if matrix size is valid*

    if (m <= 2 || m >= 10 || n <= 2 || n >= 10) {

      System.out.println("MATRIX SIZE OUT OF RANGE.");

      return;

    }

*// Initialize matrix*

    int[][] a = new int[m][n];

*// Input matrix elements*

    System.out.println("ENTER ELEMENTS OF MATRIX:");

    for (int i = 0; i < m; i++) {

      System.out.println("ENTER ELEMENTS OF ROW " +(i+1)+ ":");

      for (int j = 0; j < n; j++) {

        a[i][j] = in.nextInt();

      }

    }

*// Display the original matrix*

    System.out.println("ORIGINAL MATRIX");

    for (int i = 0; i < m; i++) {

      for (int j = 0; j < n; j++) {

        System.out.print(a[i][j] + " ");

      }

      System.out.println();

    }

*// Sorting each row using bubble sort*

    for (int i = 0; i < m; i++) {

      for (int j = 0; j < n - 1; j++) {

        for (int k = 0; k < n - j - 1; k++) {

          if (a[i][k] > a[i][k + 1]) {

*// Swap a[i][k] and a[i][k + 1]*

            int t = a[i][k];

            a[i][k] = a[i][k + 1];

            a[i][k + 1] = t;

          }

        }

      }

    }

*// Display matrix after sorting rows*

    System.out.println("MATRIX AFTER SORTING ROWS");

    for (int i = 0; i < m; i++) {

      for (int j = 0; j < n; j++) {

        System.out.print(a[i][j] + " ");

      }

      System.out.println();

    }

  }

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| **m** | **int** | Stores the number of rows in the matrix, input by the user. |
| **n** | **int** | Stores the number of columns in the matrix, input by the user. |
| **a** | **int[][]** | 2D array to hold matrix elements. |
| **t** | **int** | Temporary variable used for swapping elements during sorting. |
| **i** | **int** | Loop counter for rows. |
| **j** | **int** | Loop counter for columns. |
| **k** | **int** | Inner loop counter for bubble sort iteration within each row. |

**Question 3**

The names of the teams participating in a competition should be displayed on a banner vertically, to accommodate as many teams as possible in a single banner. Design a program to accept the names of N teams, where 2 < N < 9 and display them in vertical order, side by side with a horizontal tab (i.e. eight spaces).

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

**Example 1**

**INPUT:**  
N = 3  
Team 1: Emus  
Team 2: Road Rols  
Team 3: Coyote

**OUTPUT:**  
E R C  
m o o  
u a y  
s d o  
 t  
 R e  
 o          
 l          
 s

**Example 2**

**INPUT:**  
N = 4  
Team 1: Royal  
Team 2: Mars  
Team 3: De Rose  
Team 4: Kings

**OUTPUT:**  
R M D K  
o a e i  
y r n  
a s R g  
l o      s  
 s        
 e

**Example 3**

**INPUT:**  
N = 10

**OUTPUT:**  
INVALID INPUT

**Algorithm:**

1. **Input Validation:**
   * Accept the number N of teams participating in the competition.
   * Validate that N is greater than 2 and less than 9. If not, print "INVALID INPUT" and terminate the program.
2. **Input Team Names:**
   * Create a string array teams[ ] to store the names of the teams.
   * Use a loop to allow the user to input the names of all N teams.
   * Track the length of the longest team name (highLen).
3. **Display Teams Vertically:**
   * For each index i from 0 to highLen - 1:
     + For each team, print the i-th character. If the current team name length is less than i, print a space followed by a tab.
     + After printing the characters of all teams at the current index i, move to the next line.

**Solution:**

import java.util.Scanner;

public class Banner {

  public static void main(String *args*[]) {

    Scanner in = new Scanner(System.in);

*// Input number of teams (n)*

    System.out.print("ENTER THE VALUE OF N: ");

    int n = in.nextInt();

    in.nextLine(); *// Clear the buffer*

    in.close();

*// Check if the number of teams is within the valid range*

    if (n <= 2 || n >= 9) {

      System.out.println("INVALID INPUT");

      return;

    }

*// Initialize array to store team names*

    String[] teams = new String[n];

    int highLen = 0;

*// Input team names*

    for (int i = 0; i < n; i++) {

      System.out.print("Team " + (i + 1) + ": ");

      teams[i] = in.nextLine();

*// Track the length of the longest team name*

      if (teams[i].length() > highLen) {

        highLen = teams[i].length();

      }

    }

*// Print teams vertically side by side with tab separation*

    for (int i = 0; i < highLen; i++) {

      for (int j = 0; j < n; j++) {

        int len = teams[j].length();

        if (i >= len) {

*// If the current character index exceeds the team   
 // name length, print space with tab*

          System.out.print(" \t");

        } else {

*// Print the i-th character with tab separation*

          System.out.print(teams[j].charAt(i) + "\t");

        }

      }

      System.out.println();

    }

  }

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| **n** | **int** | Stores the number of teams participating. |
| **teams[]** | **String[]** | Array to store the names of the teams. |
| **highLen** | **int** | Stores the length of the longest team name. |
| **i** | **int** | Loop counter to traverse through the characters vertically. |
| **j** | **int** | Loop counter to traverse through the team names. |
| **len** | **int** | Stores the length of the current team name being processed. |

**Question 1**

A company manufactures packing cartons in four sizes, i.e. cartons to accommodate 6 boxes, 12 boxes, 24 boxes and 48 boxes. Design a program to accept the number of boxes to be packed (N) by the user (maximum up to 1000 boxes) and display the break-up of the cartons used in descending order of capacity (i.e. preference should be given to the highest capacity available, and if boxes left are less than 6, an extra carton of capacity 6 should be used.)

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

**Example 1**

**INPUT:**  
N = 726

**OUTPUT:**  
48 \* 15 = 720  
6 \* 1 = 6  
Remaining boxes = 0  
Total number of boxes = 726  
Total number of cartons = 16

**Example 3**

**INPUT:**  
N = 4296

**OUTPUT:**  
INVALID INPUT

**Example 2**

**INPUT:**  
N = 140

**OUTPUT:**  
48 \* 2 = 96  
24 \* 1 = 24  
12 \* 1 = 12  
6 \* 1 = 6  
Remaining boxes = 2 \* 1 = 2  
Total number of boxes = 140  
Total number of cartons = 6

**Algorithm:**

1. **Input**: Read the number of boxes N to be packed.
2. **Validation**:
   * If N is less than 1 or greater than 1000, print "INVALID INPUT" and terminate the program.
3. **Carton Sizes**: Create an array with the available carton sizes [48, 24, 12, 6].
4. **Initialize Variables**:
   * Initialize a variable total to store the total number of cartons used.
   * Initialize t to store the number of boxes left to be packed.
5. **Determine Carton Count**:
   * For each carton size in descending order, calculate how many cartons of that size are needed using integer division (t / carton\_size).
   * Subtract the packed boxes from the total boxes using the modulo operator (t = t **%** carton\_size).
   * Add the number of cartons used to the total.
6. **Remaining Boxes**:
   * If there are any boxes left (t != 0), print the remaining boxes, assign one additional carton of size 6, and increment total by 1.
7. **Output**:
   * Print the total number of boxes.
   * Print the total number of cartons used.

**Solution:**

import java.util.Scanner;

public class CartonBoxes {

  public static void main(String *args*[]) {

    Scanner in = new Scanner(System.in);

    System.out.print("Enter number of boxes (N): ");

    int n = in.nextInt();

    in.close();

    if (n < 1 || n > 1000) {

      System.out.println("INVALID INPUT");

      return;

    }

    int cartonSizes[] = { 48, 24, 12, 6 };

    int total = 0;

    int t = n;

*// loop through each carton size*

    for (int i = 0; i < cartonSizes.length; i++) {

*// find number of cartons needed for this size*

      int cartonCount = t / cartonSizes[i];

      t = t % cartonSizes[i];

      total += cartonCount;

      if (cartonCount != 0) {

        System.out.println(cartonSizes[i] + " \* " +  
 cartonCount + " = " + (cartonSizes[i] \* cartonCount));

      }

    }

*// 1 carton of capacity 6 needed if boxes left < 6*

    if (t != 0) {

      System.out.println("Remaining boxes = " + t +

" \* 1 = " + t);

      total++;

    } else {

      System.out.println("Remaining boxes = 0");

    }

    System.out.println("Total number of boxes = " + n);

    System.out.println("Total number of cartons = " + total);

  }

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data type** | **Description** |
| **n** | **int** | Stores the number of boxes to be packed (input). |
| **cartonSizes** | **int[]** | Array containing the sizes of cartons  [48, 24, 12, 6]. |
| **total** | **int** | Stores the total number of cartons used. |
| **t** | **int** | Tracks the remaining boxes to be packed. |
| **cartonCount** | **int** | Temporary variable to store how many cartons of each size are needed. |

**Question 2**

The result of a quiz competition is to be prepared as follows:

The quiz has five questions with four multiple choices (A, B, C, D), with each question carrying 1 mark for the correct answer. Design a program to accept the number of participants N such that N must be greater than 3 and less than 11. Create a double-dimensional array of size (Nx5) to store the answers of each participant row-wise. Calculate the marks for each participant by matching the correct answer stored in a single-dimensional array of size 5. Display the scores for each participant and also the participant(s) having the highest score.

Example: If the value of N = 4, then the array would be:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Q1** | **Q2** | **Q3** | **Q4** | **Q5** |
| **Participant 1** | A | B | B | C | A |
| **Participant 2** | D | A | D | C | B |
| **Participant 3** | A | A | B | A | C |
| **Participant 4** | D | C | C | A | B |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key to the question:** | D | C | C | B | A |

**Note: Array entries are line fed (i.e. one entry per line)**

Test your program for the following data and some random data.

**Example 1**

**INPUT:**  
N = 5  
Participant 1: D A B C C  
Participant 2: A A D C B  
Participant 3: B A C D B  
Participant 4: D A D C B  
Participant 5: B C A D D  
Key: B C D A A

**OUTPUT:**  
Scores:  
Participant 1 = 0  
Participant 2 = 1  
Participant 3 = 1  
Participant 4 = 1  
Participant 5 = 2  
Highest Score:  
Participant 5

**Example 3**

**INPUT:**  
N = 12

**OUTPUT:**  
INPUT SIZE OUT OF RANGE.

**Example 2**

**INPUT:**  
N = 4  
Participant 1: A C C B D  
Participant 2: B C A A C  
Participant 3: B C B A A  
Participant 4: C C D D B  
Key: A C D B B

**OUTPUT:**  
Scores:  
Participant 1 = 3  
Participant 2 = 1  
Participant 3 = 1  
Participant 4 = 3  
Highest Score:  
Participant 1  
Participant 4

**Algorithm:**

1. **Input N**: Accept the number of participants N  
    (must be between 4 and 10).
2. **Validation**: If N is less than 4 or greater than 10, print "INPUT  
    SIZE OUT OF RANGE" and terminate the program.
3. **Initialize Arrays**:
   * Create a 2D array answers of size N x 5 to store the participants' answers.
   * Create a 1D array key of size 5 to store the correct answers.
4. **Accept Answers**:
   * For each participant, accept their answers to the 5 questions and store them in the corresponding row of the answers array.
5. **Accept Answer Key**: Read the correct answers into   
    the key array.
6. **Score Calculation**:
   * Initialize an array score to store the marks for each participant.
   * Compare each participant's answers with the answer key. For each correct answer, increment the participant's score.
   * Track the highest score using a variable hScore.
7. **Output Scores**:
   * Print the score of each participant after calculating it.
8. **Determine and Display Highest Score**:
   * After scoring all participants, identify and print the participants who have the highest score.

**Solution:**

import java.util.Scanner;

public class QuizCompetition {

  public static void main(String[] *args*) {

    Scanner in = new Scanner(System.in);

    System.out.print("N = ");

    int n = in.nextInt();

*// Input validation: N should be between 4 and 10*

    if (n <= 3 || n >= 11) {

      System.out.println("INPUT SIZE OUT OF RANGE.");

      in.close();

      return;

    }

    char[][] answers = new char[n][5];

    char[] key = new char[5];

*// Accept answers for each participant*

    for (int i = 0; i < n; i++) {

      System.out.print("Participant " + (i + 1) + ": ");

      for (int j = 0; j < 5; j++) {

        answers[i][j] = in.next().charAt(0);

      }

    }

*// Accept the correct answers (Answer Key)*

    System.out.print("Key: ");

    for (int i = 0; i < 5; i++) {

      key[i] = in.next().charAt(0);

    }

    in.close();

    int[] scores = new int[n];

    int highestScore = 0;

*// Calculate the score for each participant*

    System.out.println("Scores:");

    for (int i = 0; i < n; i++) {

*// Initialize the score for each participant*

      scores[i] = 0;

      for (int j = 0; j < 5; j++) {

        if (answers[i][j] == key[j]) {

          scores[i]++;

        }

      }

      if (scores[i] > highestScore) {

        highestScore = scores[i];

      }

      System.out.println("Participant " + (i + 1) + " = " + scores[i]);

    }

*// Find and display participants with the highest score*

    System.out.println("Highest Score:");

    for (int i = 0; i < n; i++) {

      if (scores[i] == highestScore) {

        System.out.println("Participant " + (i + 1));

      }

    }

  }

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Datatype** | **Description** |
| n | int | Stores the number of participants (input). |
| answers | char[][] | 2D array storing participants' answers for each question. |
| key | char[] | 1D array storing the correct answers for the quiz. |
| score | int[] | Array storing the scores of each participant. |
| hScore | int | Stores the highest score among participants. |
| i, j | int | Loop variables for iterating over participants and answers. |

**Question 3**

Caesar Cipher is an encryption technique which is implemented as ROT13 ('rotate by 13 places'). It is a simple letter substitution cipher that replaces a letter with the letter 13 places after it in the alphabets, with the other characters remaining unchanged.

**ROT13**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A a | B b | C c | D d | E e | F f | G g | H h | I i | J j | K k | L l | M m |
| ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ | ↕ |
| N n | O o | P p | Q q | R r | S s | T t | U u | V v | W w | X x | Y y | Z z |

Write a program to accept a plain text of length L, where L must be greater than 3 and less than 100.

Encrypt the text if valid as per the Caesar Cipher.

Test your program with the sample data and some random data

**Example 1**

**INPUT:**  
Hello! How are you?

**OUTPUT:**  
The cipher text is:  
Uryyb! Ubj ner lbh?

**Example 2**

**INPUT:**  
Encryption helps to secure data.

**OUTPUT:**  
The cipher text is:  
Rapelcgvba urycf gb frpher qngn.

**Example 3**

**INPUT:**  
You

**OUTPUT:**  
INVALID LENGTH

**Algorithm:**

1. **Input plain text**: Read a string (plain text) from the user.
2. **Check text length**: Ensure the length of the text is greater than 3 and less than 100. If not, display "INVALID LENGTH" and terminate.
3. **Initialize result string**: Prepare an empty string to store the encrypted text.
4. **Loop through each character in the plain text**:
   * If the character is an uppercase letter between 'A' and 'M' or lowercase between 'a' and 'm', add 13 to its ASCII value.
   * If the character is an uppercase letter between 'N' and 'Z' or lowercase between 'n' and 'z', subtract 13 from its ASCII value.
   * If the character is not a letter (non-alphabetic), leave it unchanged.
5. **Store encrypted characters**: Append each processed character to the result string.
6. **Output the cipher text**: Print the final encrypted string.

**Solution:**

import java.util.Scanner;

public class CaesarCipher {

  public static void main(String[] *args*) {

    Scanner in = new Scanner(System.in);

    System.out.println("Enter plain text:");

    String str = in.nextLine();

    in.close();

    int len = str.length();

    if (len <= 3 || len >= 100) { *// Validate length*

      System.out.println("INVALID LENGTH");

      return;

    }

*// Initialize an empty string to store cipher text*

    String cipher = "";

*// Loop through each character in the plain text*

    for (int i = 0; i < len; i++) {

      char ch = str.charAt(i);

      if ((ch >= 'A' && ch <= 'M') ||

(ch >= 'a' && ch <= 'm')) {

*// Rotate by +13 for A-M or a-m*

        cipher += (char) (ch + 13);

      } else if ((ch >= 'N' && ch <= 'Z') ||

(ch >= 'n' && ch <= 'z')) {

*// Rotate by -13 for N-Z or n-z*

        cipher += (char) (ch - 13);

      } else {

*// Leave non-alphabetic characters unchanged*

        cipher += ch;

      }

    }

    System.out.println("The cipher text is:\n" + cipher);

  }

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| str | String | Stores the user input plain text |
| len | int | Length of the input plain text |
| ch | char | Holds each character of the string while processing |
| i | int | Loop counter to iterate through each character of the string |
| cipher | String | The final encrypted cipher text |
| sb | String | The encrypted result as it's being constructed |

**Question 7**

Design a class Convert to find the date and the month from a given day number for a particular year.

**Example:** If day number is 64 and the year is 2020, then the  
 corresponding date would be:

March 4, 2020 i.e. (31 + 29 + 4 = 64)

Some of the members of the class are given below:

**Class name:** Convert

**Data members/instance variables:**

n : integer to store the day number

d : integer to store the day of the month (date)

m : integer to store the month

y : integer to store the year

**Methods/Member functions:**

Convert() : constructor to initialize the data members  
 with legal initial values.

void accept() : to accept the day number and the year.

void day\_to\_date(): converts the day number to its corresponding  
 date for a particular year and stores the date  
 in ‘d’ and the month in ‘m’.

void display() : displays the month name, date and year.

**Algorithm:**

1. **Initialize Variables:**
   * Create a class Convert with instance variables: n (day number), d (day of the month), m (month), and y (year).
2. **Input:**
   * In the accept() method, take user input for the day number (n) and the year (y).
3. **Determine Leap Year:**
   * In the day\_to\_date() method, check if the given year is a leap year. A leap year occurs if:

The year is divisible by 4, but not divisible by 100, unless divisible by 400.

1. **Set Month Boundaries:**
   * Define arrays for the number of days in each month for leap and non-leap years.
   * Use a loop to find the corresponding month by subtracting the number of days in each month from n until n becomes less than the number of days in that month.
2. **Determine Date:**
   * The remaining value of n will be the date, and the month will be the index of the loop.
3. **Display Result:**
   * In the display() method, map the month number to its name and print the date, month name, and year.

**Solution:**

import java.util.Scanner;

public class Convert {

*// Instance variables*

  int n, d, m, y;

*// Constructor to initialize data members*

  Convert() {

n = 0; d = 0; m = 0; y = 0;

  }

*// Method to accept day number and year*

  void accept() {

    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the day number: ");

    n = sc.nextInt();

    System.out.print("Enter the year: ");

    y = sc.nextInt();

    sc.close();

  }

*// Method to check if the year is a leap year*

  boolean isLeapYear(int *year*) {

    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

  }

*// Method to convert day number to month and date*

  void day\_to\_date() {

*// Days in months for non-leap years and leap years*

    int[] daysInMonths = { 31, 28, 31, 30, 31, 30,

31, 31, 30, 31, 30, 31 };

// Update February days if leaf year

    daysInMonths[1] = isLeapYear(y) ? 1 : 0;

*// Determine the month and date*

    int remainingDays = n;

    for (int i = 0; i < 12; i++) {

      if (remainingDays <= daysInMonths[i]) {

        m = i + 1; *// Month number*

        d = remainingDays; *// Day of the month*

        break;

      } else {

        remainingDays -= daysInMonths[i];

      }

    }

  }

*// Method to display the date in month name format*

  void display() {

*// Month names array*

    String[] monthNames = {

        "January", "February", "March", "April",

"May", "June", "July", "August",

"September", "October", "November", "December"

    };

*// Display the result*

    System.out.println(monthNames[m-1] + " " + d + ", " + y);

  }

*// Main method to run the program*

  public static void main(String[] *args*) {

    Convert obj = new Convert();

    obj.accept();

    obj.day\_to\_date();

    obj.display();

  }

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| n | int | Stores the day number provided by the user. |
| d | int | Stores the day of the month (calculated). |
| m | int | Stores the month number (calculated). |
| y | int | Stores the year provided by the user. |
| daysInMonths | int[] | Array to hold the number of days in each month for a given year type (leap or non-leap). |
| monthNames | String[] | Array to hold month names for display. |

**Question 10**

A Circular queue is a linear data structure which works on the principle of FIFO, enables the user to enter data from the rear end and remove data from the front end with the rear end connected to the front end to form a circular pattern.

Define a class CirQueue with the following details:

**Class name:** CirQueue

**Data members / instance variables:**

cq[ ] : array to store the integers

cap : stores the maximum capacity of the array

front : to point the index of the front end

rear : to point the index of the rear end

**Member functions:**

CirQueue(int max): constructor to initialize the data member  
 cap=max, front=0 and rear=0.

void push(int n) : to add integer in the queue from the  
 rear end if possible, otherwise  
 display the message “QUEUE IS FULL”.

int pop() : removes and returns the integer  
 from the front end of the queue if  
 any, else returns -9999.

void show() : displays the queue elements.

**Solution:**

import java.util.Scanner;

public class CirQueue {

*// Instance variables*

  int[] cq;

  int cap;

  int front;

  int rear;

*// Constructor to initialize queue with given capacity*

  CirQueue(int *max*) {

    cap = max;

    cq = new int[cap];

    front = 0;

    rear = 0;

  }

*// Method to add an integer to the queue (enqueue)*

  void push(int *n*) {

    if ((rear + 1) % cap == front) {

      System.out.println("QUEUE IS FULL");

    } else {

*// Add to rear position*

      cq[rear] = n;

*// Increment rear circularly*

      rear = (rear + 1) % cap;

    }

  }

*// Method to remove and return integer from queue (dequeue)*

  int pop() {

    if (front == rear) {

*// Queue is empty*

      return -9999;

    } else {

*// Retrieve front element*

      int val = cq[front];

*// Increment front circularly*

      front = (front + 1) % cap;

      return val;

    }

  }

*// Method to display the elements of the queue*

  void show() {

    if (front == rear) {

      System.out.println("QUEUE IS EMPTY");

    } else {

      System.out.print("Queue elements: ");

      int i = front;

      while (i != rear) {

        System.out.print(cq[i] + " ");

*// Circular increment*

        i = (i + 1) % cap;

      }

      System.out.println();

    }

  }

*// Main method to run the program*

  public static void main(String[] *args*) {

    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the capacity of the circular queue: ");

    int max = sc.nextInt();

    CirQueue queue = new CirQueue(max);

    while (true) {

      System.out.println("\n1. Push\n2. Pop\n3. Show\n4. Exit");

      System.out.print("Enter your choice: ");

      int choice = sc.nextInt();

System.out.println();

      switch (choice) {

        case 1:

          System.out.print("Enter an integer to push: ");

          int value = sc.nextInt();

          queue.push(value);

          break;

        case 2:

          int removedValue = queue.pop();

          if (removedValue == -9999) {

            System.out.println("QUEUE IS EMPTY");

          } else {

            System.out.println("Popped value: " + removedValue);

          }

          break;

        case 3:

          queue.show();

          break;

        case 4:

          sc.close();

          System.exit(0);

        default:

          System.out.println("Invalid choice, try again.");

      }

    }

  }

}

**Variable Data Chart:**

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| cq[] | int[] | Array to store the integers in the circular queue. |
| cap | int | Stores the maximum capacity of the circular queue array. |
| front | int | Index that points to the front end of the queue (element to be dequeued next). |
| rear | int | Index that points to the rear end of the queue (position to enqueue the next element). |
| n | int | Input integer to be added to the queue in the push() method. |
| max | int | Input value for the maximum capacity of the circular queue, passed to the constructor. |
| val | int | Temporary variable used to store the dequeued integer in the pop() method. |
| choice | int | Variable used in the main() method to store the user's menu choice. |
| i | int | Loop counter used in the show() method to iterate through the queue elements for display. |
| removedValue | int | Stores the value returned by the pop() method to check if the queue was empty. |

**Question 9**

A class Mix has been defined to mix two words, character by character, in the following manner:

The first character of the first word is followed by the first character of the second word and so on. If the words are of different length, the remaining characters of the longer word are put at the end.

Example: If the First word is “JUMP” and the second word is “STROLL”, then the required word will be “JSUTMRPOLL”

Some of the members of the class are given below:

**Class name:** Mix

**Data member/instance variable:**

wrd : to store a word

len : to store the length of the word

**Member functions/methods:**

Mix() : default constructor to initialize the  
 data members with legal initial values.

void feedword() : to accept the word in UPPER case.

void mix\_word(Mix P, Mix Q): mixes the words of objects P and Q  
 as stated above and stores the  
 resultant word in the current object.

void display() : displays the word.

**Algorithm:**

1. **Initialize Variables:**
   * Define the class Mix with two instance variables: wrd (to store the word) and len (to store the length of the word).
   * Create a constructor to initialize wrd and len with default values.
2. **Feed Word:**
   * In the feedword() method, accept a word in uppercase and assign it to wrd.
   * Calculate and store the length of the word in len.
3. **Mix Word:**
   * Define the mix\_word() method which takes two Mix objects (P and Q) as parameters.
   * Loop through the characters of both words:
     + Add one character from P.wrd followed by one character from Q.wrd to the resultant word.
   * If the words are of different lengths, append the remaining characters of the longer word.
   * Store the resultant word in the current object.
4. **Display:**
   * Define the display() method to print the mixed word.