Question 1:

Today in class, Kiran read about how to find the factorial of a number and did a great job of understanding it, while her classmate Seema struggled. Write a code in C language which will help Seema to find out the factorial of the number.

Sample Input 1:

5

Sample Output 1:

120

Sample Input 2:

7

Sample Output 2:

5040

Input Explanation:

The user is prompted to enter a positive integer for which they want to calculate the factorial.

Output Explanation:

The program calculates the factorial of the input number and prints the result.

The input number and the resulting factorial are frequently included in the output message.

Test cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**0**	**1**	**3**	**6**
Output	**1**	**1**	**6**	**720**

Solution:

```
#include <stdio.h>

int main() {
    int n, i, fact = 1;

    // printf("Enter a non-negative integer: ");
    scanf("%d", &n);

    if (n == 0) {
        printf("Factorial of 0 is 1\n");
    } else {
        for (i = 1; i <= n; i++) {
            fact = fact * i;
        }
```

```
        printf("Factorial of %d is %d\n", n, fact);
    }
     return 0;
}
```

Question 2:

As the director of a library, Radhika is responsible for buying all of the books that are added to the collection. Radhika now has to buy 10 books for her library, and she will receive a 10% discount on those 5 books. Write a C program to prepare a bill for 5 books after 10% discount for her library.

Input Explanation:

The user is prompted to enter the price of each book, one at a time, for the number of books that is 5

Output Explanation:

The program outputs the total cost of the books after the discount.

Solution:

```
#include <stdio.h>

int main() {
    int num_books=5;
    float book_price, discount = 0.1, total_cost = 0.0;
```

```
    for (int i = 0; i < num_books; i++) {
        printf("Enter the price of book %d: $", i+1);
        scanf("%f", &book_price);
        total_cost += book_price;
    }

    total_cost *= (1 - discount);

    printf("Total cost for %d books after 10%% discount: $%.2f\n",
num_books, total_cost);

    return 0;
}
```

Question 3:

Imagine if you were hosting a dinner party and invited all the numbers from 1 to n to the table, but instead of bringing their whole selves, they only brought their fractions. As the host, your task is to calculate the sum of all the fractional guests, starting with the gracious host at 1 and adding in each

party guest until the table is full. Will you be able to solve the mystery of the sum of the fractions? Write a C program to calculate the sum of following series where n is input by user.

1 + 1/2 + 1/3 + 1/4 + 1/5 +……1/n

Sample Input 1:

5

Sample Output 1:

2.28

Sample Input 2:

-1

Sample Output 2:

Invalid input

Input Explanation:

The user is prompted to enter a value for n.

The user should enter a positive integer value for n that represents the number of terms in the series to be summed.

Output Explanation:

The program will calculate the sum of series and output it with two decimal places.

Test Cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**0**	**-3**	**15**	**33**
Output	**0**	**Invalid input**	**3.32**	**4.09**

Solution:

```
#include <stdio.h>


int main() {
    int n;
    float sum = 0.0;


   // printf("Let's calculate the sum of the series 1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n.\n");
 //   printf("Enter the value of n: ");
    scanf("%d", &n);
    if(n==0)
    {
       printf("%d",0);
    }
    else if(n<0)
```

```
    {
        printf("Invalid input :(");
    }
    else{

    for (int i = 1; i <= n; i++) {
        sum += 1.0 / i;
    }
    }

    printf("The sum of the series is: %.2f\n", sum);

    return 0;
}
```

Question 4:

Would you like to explore the endless possibilities of language and discover the unique ways in which words can be arranged to convey different meanings? If so, then you'll love the challenge of writing a C program that can detect anagrams with the help of function.

An anagram is a word or phrase created by rearranging the letters of another word or phrase. In the context of the anagram code, the two input strings will be compared to see if they contain the same letters in different orders.

Sample Input 1:

hello

world

Sample output 1:

hello and world are not anagrams to each other

Sample Input 2:

dormitory

dirty room

Sample output 2:

dormitory and dirty room are anagrams to each other

Input Explanation:

Take two strings as input

Output Explanation:

If the two strings are anagrams, the output of the code will typically be a message indicating that they are anagrams.

If the two strings are not anagrams, the output of the code will typically be a message indicating that they are not anagrams.

Test cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**astronomer** **moon starer**	**aabbcc** **abcabc**	**restful** **fluster**	**schoolmaster** **the classroom**
Output	**These are anagrams**	**These are not anagrams**	**These are anagrams**	**These are anagrams**

Solution:

```
#include <stdio.h>
#include <string.h>

// Function to check if two strings are anagrams of each other
int check_anagram(char [], char []);

int main() {
    char str1[100], str2[100];

   // printf("Enter first string: ");
    gets(str1);

 //   printf("Enter second string: ");
    gets(str2);
```

```
    if (check_anagram(str1, str2) == 1)
        printf("%s and %s are anagrams of each other.\n", str1, str2);
    else
        printf("%s and %s are not anagrams of each other.\n", str1, str2);

    return 0;
}

int check_anagram(char a[], char b[]) {
    int first[26] = {0}, second[26] = {0}, c = 0;

    // Count frequency of each character in first string
    while (a[c] != '\0') {
        first[a[c]-'a']++;
        c++;
    }

    c = 0;

    // Count frequency of each character in second string
    while (b[c] != '\0') {
        second[b[c]-'a']++;
        c++;
    }
```

```
    // Compare the frequency of each character in both strings
    for (c = 0; c < 26; c++) {
        if (first[c] != second[c])
            return 0;
    }

    return 1;
}
```

Question 5:

Create a C program that takes a positive integer n as input from the user and prints a pattern of n rows, where each row contains a sequence of integers in increasing or decreasing order.

1

3 2

4 5 6

10 9 8 7

11 12 13 14 15

If a row contains an odd number of integers, they are printed in increasing order, while if a row contains an even number of integers, they are printed in decreasing order.

Input Explanation:

The program prompts the user to enter a value for n, which is the number of rows in the pattern. The user should enter a positive integer value for n. If the user enters a non-positive integer or a non-integer value, the program will not work correctly.

Output Explanation:

The program generates a pattern of n rows, where each row contains a sequence of integers in increasing or decreasing order.

Solution:

```
#include <stdio.h>


int main() {
    int n;
   // printf("Enter the value of n: ");
    scanf("%d", &n);


    int num = 1;


    for (int i = 1; i <= n; i++) {
        if (i % 2 != 0) {
            for (int j = 1; j <= i; j++) {
                printf("%d ", num);
                num++;
            }
```

```
        } else {
            int temp = num;
            for (int j = 1; j <= i; j++) {
                printf("%d ", temp - 1);
                temp--;
            }
            num += i;
        }
        printf("\n");
    }
    return 0;
}
```

Question 6:

Create a C program that takes a positive integer n as input from the user and prints a pattern of n rows:

0

1 0

1 0 1

0 1 0 1

0 1 0 1 0

Input Explanation:

The program prompts the user to enter the number of rows for the pattern.

Output Explanation:

For each element in the pattern, the program checks if the sum of the current row number and the current column number is even or odd. If the sum is even, the program prints a "0" character, and if the sum is odd, the program prints a "1" character. Finally, the program prints a newline character to move to the next row.

Solution:

```
#include <stdio.h>


int main() {
    int n, i, j;


 //   printf("Enter the number of rows for the pattern: ");
    scanf("%d", &n);


    for (i = 1; i <= n; i++) {
        for (j = 1; j <= i; j++) {
            if ((i + j) % 2 == 0) {
                printf("0 ");
            } else {
                printf("1 ");
            }
```

```
        }
        printf("\n");
    }

    return 0;
}
```

Question 7:

Julie took a test of the students of her class today. She wants the student whose marks are second highest in the class. Write a program in C to the student who has the second highest marks in the class with the help of array.

Sample Input 1:

5

1 2 3 4 5

Sample output 1:

4

Sample Input 2:

6

2 2 2 2 2 2

Sample output 2:

There is no second largest element in the array

Input Explanation:

The program takes input from the user in two steps:

First, the program prompts the user to enter the size of the array. The user must enter an integer value that represents the number of elements in the array.

Second, the program prompts the user to enter the elements of the array. The user must enter size number of integer values separated by spaces.

Output Explanation:

The program outputs one of two possible messages:

If the array contains a second largest element, then the program outputs a message that shows the value of the second largest element in the array.

If the array does not contain a second largest element, then the program outputs a message that says there is no second largest element in the array.

Test cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**5** **5 4 3 2 1**	**4** **1 1 1 1**	**6** **8 6 9 2 7 3**	**3** **-5 -10 0**
Output	**4**	**There is no second largest element in the array**	**8**	**-5**

Solution:

```
#include <stdio.h>

void find_second_largest(int arr[], int size) {
    int largest = arr[0];
    int second_largest = arr[0];

    for (int i = 1; i < size; i++) {
        if (arr[i] > largest) {
            second_largest = largest;
            largest = arr[i];
        } else if (arr[i] > second_largest && arr[i] != largest) {
            second_largest = arr[i];
        }
    }

    if (second_largest != largest) {
        printf("The second largest element in the array is: %d", second_largest);
    } else {
        printf("There is no second largest element in the array.");
    }
}

int main() {
```

```
    int arr[100];
    int size;

  //  printf("Enter the size of the array: ");
    scanf("%d", &size);

 //   printf("Enter the elements of the array: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }

    find_second_largest(arr, size);

    return 0;
}
```

Question 8:

What if I told you that with just a little bit of string manipulation, you could rearrange the words of a phrase to give it a whole different meaning? Can you figure out how to make a message by flipping the words in a string?

Write a program in C to reverse a string.

Sample Input 1:

"Hello World"

Sample output 1:

"World Hello"

Sample Input 2:

"The quick brown fox jumps over the lazy dog"

Sample output 2:

"dog lazy the over jumps fox brown quick The"

Input Explanation:

The program takes input from the user in the following way:

The program prompts the user to enter a string.

The user must enter a string that contains one or more words. The maximum length of the string is 100 characters.

Output Explanation:

The program outputs the reversed string, which consists of the words of the original string in reverse order.

Solution:

```
#include <stdio.h>

#include <string.h>

void reverseWords(char *s) {
```

```
    char *word = strtok(s, " ");
    char *words[100];
    int count = 0;
    while (word != NULL) {
        words[count++] = word;
        word = strtok(NULL, " ");
    }
    for (int i = count - 1; i >= 0; i--) {
        printf("%s ", words[i]);
    }
}

int main() {
    char str[100];
    //printf("Enter a string: ");
    fgets(str, 100, stdin);
    reverseWords(str);
    return 0;
}
```

Question 9:

James Challenged his brother to unravel the mysteries and find the hidden connections between words and discover the longest and most unchanging

sequence of characters that tie a set of strings together. To find the common thread that weaves them into a harmonious whole.

Write a program in C to find the longest and most unchanging sequence of characters in string.

Sample Input 1:

"hydroplane"

"hydrant"

"hydrosphere"

Sample output 1:

"hydro"

Sample Input 2:

"interstellar"

"intersect"

"intertwine"

Sample output 2:

"inter"

Input Explanation:

The user to enter the number of strings.

Then, the program prompts the user to enter each string one by one.

Output Explanation:

The program outputs the longest common prefix among the input strings.

Solution:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char* longestCommonPrefix(char **strs, int n) {
    if (n == 0) {
        return "";
    }
    if (n == 1) {
        return strs[0];
    }
    int minLen = strlen(strs[0]);
    for (int i = 1; i < n; i++) {
        int len = strlen(strs[i]);
        if (len < minLen) {
            minLen = len;
        }
    }
    int i, j;
    for (i = 0; i < minLen; i++) {
        for (j = 1; j < n; j++) {
```

```
            if (strs[j][i] != strs[0][i]) {
                break;
            }
        }
        if (j != n) {
            break;
        }
    }
    char *result = malloc(sizeof(char) * (i + 1));
    strncpy(result, strs[0], i);
    result[i] = '\0';
    return result;
}

int main() {
    int n;
    //printf("Enter the number of strings: ");
    scanf("%d", &n);
    char *strs = (char)malloc(sizeof(char) * n);
    for (int i = 0; i < n; i++) {
        printf("Enter string %d: ", i+1);
        char str[100];
        scanf("%s", str);
        strs[i] = (char*)malloc(sizeof(char) * (strlen(str) + 1));
```

```
        strcpy(strs[i], str);
    }
    char *prefix = longestCommonPrefix(strs, n);
    printf("Longest common prefix: %s\n", prefix);
    free(prefix);
    for (int i = 0; i < n; i++) {
        free(strs[i]);
    }
    free(strs);
    return 0;
}
```

Question 10:

Reveal the mysteries of the Roman numeral system by travelling back in time to the Roman Empire. Are you able to unravel the mysteries of Roman numbers and translate them into modern language?

Write a program in C to convert Roman numerals to integers.

Sample Input 1:

"X"

Sample output 1:

10

Sample Input 2:

"MCMXCIV"

Sample output 2:

1994

Input Explanation:

The input to the program is a valid Roman numeral entered by the user. A Roman numeral is a system of numerical notation used in ancient Rome, in which letters of the alphabet are used to represent numbers.

The valid letters used in Roman numerals are:

I = 1

V = 5

X = 10

L = 50

C = 100

D = 500

M = 1000

The user can enter a valid Roman numeral as a string, such as "X" or "MCMXCIV", up to a maximum of 20 characters in length.

Output Explanation:

The output of the program is the decimal equivalent of the Roman numeral entered by the user, displayed as a formatted string on the console.

The output string has the following format: "The decimal equivalent of [romanNum] is [decimalNum].", where [romanNum] is the Roman numeral entered by the user and [decimalNum] is its decimal equivalent.

If the input is not valid, the program outputs an error message and exits with an error code of 1.

Test cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**“MCMLXXXIV”**	**“IC”**	**“CD”**	**ABC**
Output	**1984**	**Error: Invalid Roman numeral**	**400**	**Error: Invalid Roman numeral**

Solution:

```
#include <stdio.h>

#include <string.h>


int romanToDecimal(char romanNum[]);


int main()

{

    char romanNum[21];

    int decimalNum;
```

```
 //   printf("Enter a Roman numeral: ");
    if (scanf("%20s", romanNum) != 1) {
        printf("Error: Invalid input.\n");
        return 1;
    }

    decimalNum = romanToDecimal(romanNum);

    if (decimalNum == -1) {
        printf("Error: Invalid Roman numeral.\n");
        return 1;
    }

    printf("The decimal equivalent of %s is %d.\n", romanNum, decimalNum);
    return 0;
}

int romanToDecimal(char romanNum[])
{
    int i, decimalNum = 0, prevValue = 1000, curValue;

    for (i = 0; i < strlen(romanNum); i++) {
        switch (romanNum[i]) {
            case 'I':
                curValue = 1;
```

```
            break;
        case 'V':
            curValue = 5;
            break;
        case 'X':
            curValue = 10;
            break;
        case 'L':
            curValue = 50;
            break;
        case 'C':
            curValue = 100;
            break;
        case 'D':
            curValue = 500;
            break;
        case 'M':
            curValue = 1000;
            break;
        default:
            return -1;
    }

    if (curValue > prevValue) {
        decimalNum += curValue - 2 * prevValue;
```

```
        } else {
            decimalNum += curValue;
        }

        prevValue = curValue;
    }

    return decimalNum;
}
```

Question 11:

Aditya was tasked by Abhinav to demonstrate his programming skills by creating a software that can differentiate between the strong consonants and the melodic vowels in a string of letters. Can you accurately count how many there are?

Write a C program to help Aditya.

Sample Input 1:

"Hello World"

Sample output 1:

Number of vowels: 3

Number of consonants: 7

Sample Input 2:

"The quick brown fox jumps over the lazy dog"

Sample output 2:

Number of vowels: 9

Number of consonants: 19

Input Explanation:

The user is prompted to enter a string of characters. The maximum length of the input string is 100 characters.

Output Explanation:

The program outputs the number of vowels and consonants in the input string. The output is in the format "Number of vowels: X" and "Number of consonants: Y", where X is the number of vowels and Y is the number of consonants.

Test cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**“1234”**	**“AaEeIiOoUu”**	**apple**	**“a e i o u”**
Output	Number of vowels: 0 Number of consonants: 0	Number of vowels: 10 Number of consonants: 0	Number of vowels: 2 Number of consonants: 3	Number of vowels: 5 Number of consonants: 0

Solution:

```
#include <stdio.h>

#include <string.h>
```

```
#include <ctype.h>

int main() {
    char str[100];
    int num_vowels = 0;
    int num_consonants = 0;
   // printf("Enter a string: ");
    fgets(str, 100, stdin);
    for (int i = 0; i < strlen(str); i++) {
        if (isalpha(str[i])) {
            switch(tolower(str[i])) {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':
                    num_vowels++;
                    break;
                default:
                    num_consonants++;
                    break;
            }
        }
    }
```

```
    printf("Number of vowels: %d\nNumber of consonants: %d\n", num_vowels,
num_consonants);
    return 0;
}
```

Question 12:

Let's take on a challenge in the world of pointers in C programming! Your goal is to write a program that effectively uses pointers to access the values and addresses of variables.

Sample Input 1:

42

Sample output 1:

Value of num: 42

Address of num: 0x7ffc45ed3a94

Value of ptr: 0x7ffc45ed3a94

Value pointed to by ptr: 42

Input Explanation:

The program prompts the user to enter a number. The user should enter an integer value when prompted.

Output Explanation:

The program outputs the following information:

The value of num: This is the value that the user entered.

The address of num: This is the memory address where num is stored.

The value of ptr: This is the value of the pointer variable ptr, which is the memory address where num is stored.

The value pointed to by ptr: This is the value that the pointer ptr is pointing to, which is the same as the value of num.

Solution:

```
#include <stdio.h>


int main() {
  int num;
  int *ptr;

  printf("Enter a number: ");
  scanf("%d", &num);

  ptr = &num;

  printf("Value of num: %d\n", num);
  printf("Address of num: %p\n", &num);
  printf("Value of ptr: %p\n", ptr);
  printf("Value pointed to by ptr: %d\n", *ptr);

  return 0;
```

}

Question 13:

Unlock the secrets of string length with the help of pointers! In this challenge, you must write a C program that finds the number of characters in a given string. Using pointers, you will traverse the string until the end, marked by the null character ('\0'), counting each character along the way. Embark on this exciting journey to measure the magnitude of strings with the power of pointers!

Sample Input 1:

"hello"

Sample output 1:

5

Sample Input 2:

"greeting"

Sample output 2:

8

Input Explanation:

A string of characters.

Output Explanation:

An integer indicating the number of characters in the input string, excluding the null character ('\0').

Test cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**“Hello, world!”**	**“12345”**	**“”(empty string)**	**“Qwerty”**
Output	**13**	**5**	**0**	**6**

Solution:

```
#include <stdio.h>


int main() {
    char str[100];
    int len = 0;
    char *ptr;

  //  printf("Enter a string: ");
    fgets(str, 100, stdin);


    ptr = str;


    while (*ptr != '\0') {
        len++;
        ptr++;
```

```
    }

    printf("Length of string: %d", len);

    return 0;
}
```

Question 14:

Write a program in C that demonstrates the use of different storage classes and their scope. The program should define variables with the following storage classes:

auto

static

extern

register

The program should also print the values of these variables after defining them, and then modify their values in a function and print their values again to demonstrate the scope and lifetime of these variables.

Output Explanation:

The program outputs the values of the different variables after defining them, and then the values again after modifying them. The output shows the effect of the different storage classes and their scope and lifetime.

Solution:

```
#include <stdio.h>
```

```
void modify_values();

int main()
{
    auto int auto_var = 1;
    static int static_var = 2;
    extern int extern_var;
    register int register_var = 3;

//    printf("Values after defining:\n");
    printf("auto_var = %d\n", auto_var);
    printf("static_var = %d\n", static_var);
    printf("extern_var = %d\n", extern_var);
    printf("register_var = %d\n", register_var);

    modify_values();

    printf("Values after modifying:\n");
    printf("auto_var = %d\n", auto_var);
    printf("static_var = %d\n", static_var);
    printf("extern_var = %d\n", extern_var);
    printf("register_var = %d\n", register_var);

    return 0;
```

```
}

void modify_values()
{
    auto int auto_var = 4;
    static int static_var = 5;
    extern int extern_var = 6;
    register int register_var = 7;

    printf("Values inside modify_values:\n");
    printf("auto_var = %d\n", auto_var);
    printf("static_var = %d\n", static_var);
    printf("extern_var = %d\n", extern_var);
    printf("register_var = %d\n", register_var);
}
```

Solution Explanation:

Auto variable x before modification: 2

Static variable y before modification: 3

Extern variable z before modification: 4

Register variable w before modification: 5

Inside modify_variables function:

Auto variable x after modification: 10

Static variable y after modification: 20

Extern variable z after modification: 30

Register variable w after modification: 40

After modify_variables function:

Auto variable x after modification: 2

Static variable y after modification: 23

Extern variable z after modification: 30

Register variable w after modification: 5

In this example, the program defines variables with different storage classes, prints their initial values, and then modifies their values in a function. The function demonstrates how the different storage classes affect the scope and lifetime of the variables, and the output shows the results of the modifications.

Question 15:

Create a C program that uses pointers to swap the values of two integer variables. Two integers should be entered by the user into the application, which should then use pointer variables to swap the values. The values of the two variables before and after the swap should then be printed by the software.

Sample Input 1:

5, 7

Sample output 1:

Before swap: a=5, b=7

After swap: a=7, b=5

Sample Input 2:

12, 15

Sample output 2:

Before swap: a=12, b=15

After swap: a=15, b=12

Input Explanation:

The program prompts the user to enter two integer values.

Output Explanation:

The program prints the values of the two integers to show that their values have been swapped.

Test cases:

	Test case 1	Test case 2	Test case 3	Test case 4
Input	**-8, 0**	**1, 1**	**10, 20**	**0, 0**
Output	Before swap: a=-8, b=0 After swap: a=0, b=-8	Before swap: a=1, b=1 After swap: a=1, b=1	Before swap: a=10 b=20 After swap: a=20, b=10	Before swap: a=0, b=0 After swap: a=0, b=0

Solution:

```
#include <stdio.h>

int main() {
    int a, b, temp;
    int *ptr1, *ptr2;

 //   printf("Enter two integers:\n");
    scanf("%d %d", &a, &b);

    printf("Before swap: a = %d, b = %d\n", a, b);

    ptr1 = &a;
    ptr2 = &b;

    temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;

    printf("After swap: a = %d, b = %d\n", a, b);

    return 0;
```

}

.