**Accenture Coding Questions**

The following is a compilation ofthe type of **Accenture coding questions** that you’d be encountering.

**######################################################################################################################################################**

**1. Execute the given function.**

def differenceofSum(n.m)

The function takes two integrals m and n as arguments. You are required to obtain the total of all integers ranging between 1 to n (both inclusive) which are not divisible by m. You must also return the distinction between the sum of integers not divisible by m with the sum of integers divisible by m.

Assumption

o m > 0 and n > 0

o Sum lies within the integral range

**Example**

Input:  
 m = 6  
 n = 30

Output:  
 285

o Integers divisible by 6 are 6, 12, 18, 24, and 30. Their sum is 90.

o Integers that are not divisible by 6 are 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 25, 26, 27, 28, and 29. Their sum is 375.

o The difference between them is 285 (375 – 90).

Sample input:  
 m = 3  
 n = 10

Sample output:  
 19

we need to write a C program that performs the following:

1. **Calculate the Sum of All Integers from 1 to n**: This sum is given by the formula S=n×(n+1)2S = \frac{n \times (n + 1)}{2}S=2n×(n+1)​.
2. **Calculate the Sum of Integers Divisible by m**: To do this, we need to find how many integers from 1 to n are divisible by m, and then sum them up. If the count of such integers is kkk, then the integers divisible by m form an arithmetic series: m,2m,3m,…,kmm, 2m, 3m, \ldots, kmm,2m,3m,…,km. The sum of this series can be computed using:

Sum of multiples of m=m×(k×(k+1)2)\text{Sum of multiples of } m = m \times \left(\frac{k \times (k + 1)}{2}\right)Sum of multiples of m=m×(2k×(k+1)​)

where kkk is the largest integer such that k×m≤nk \times m \leq nk×m≤n. kkk can be calculated as k=⌊nm⌋k = \left\lfloor \frac{n}{m} \right\rfloork=⌊mn​⌋.

1. **Calculate the Sum of Integers Not Divisible by m**: This can be obtained by subtracting the sum of integers divisible by m from the total sum of integers from 1 to n.
2. **Calculate the Distinction**: The distinction is the difference between the sum of integers not divisible by m and the sum of integers divisible by m.

Here’s the C program to perform these calculations:

**#include <stdio.h>**

**int sum\_of\_integers(int n) {**

**return n \* (n + 1) / 2;**

**}**

**int sum\_of\_multiples(int m, int n) {**

**int k = n / m;**

**return m \* k \* (k + 1) / 2;**

**}**

**int main() {**

**int m, n;**

**// Input values for m and n**

**printf("Enter the value of m: ");**

**scanf("%d", &m);**

**printf("Enter the value of n: ");**

**scanf("%d", &n);**

**// Calculate the total sum of integers from 1 to n**

**int total\_sum = sum\_of\_integers(n);**

**// Calculate the sum of integers divisible by m**

**int sum\_divisible\_by\_m = sum\_of\_multiples(m, n);**

**// Calculate the sum of integers not divisible by m**

**int sum\_not\_divisible\_by\_m = total\_sum - sum\_divisible\_by\_m;**

**// Calculate the distinction**

**int distinction = sum\_not\_divisible\_by\_m - sum\_divisible\_by\_m;**

**// Output the results**

**printf("Sum of integers not divisible by %d: %d\n", m, sum\_not\_divisible\_by\_m);**

**printf("Sum of integers divisible by %d: %d\n", m, sum\_divisible\_by\_m);**

**printf("Distinction: %d\n", distinction);**

**return 0;**

**}**

**Explanation of the Code:**

1. **sum\_of\_integers(int n)**: Computes the sum of all integers from 1 to n using the formula.
2. **sum\_of\_multiples(int m, int n)**: Computes the sum of all multiples of m up to n.
3. **main()**:
   * Reads the values of m and n.
   * Computes the total sum of integers from 1 to n.
   * Computes the sum of integers divisible by m.
   * Calculates the sum of integers not divisible by m.
   * Computes the distinction between the sum of integers not divisible by m and the sum of integers divisible by m.
   * Prints the results.

This program will give you the required sums and distinctions based on user input.

**######################################################################################################################################################**

**2. Execute the given function.**

def LargeSmallSum(arr)

The function takes an integral arr which is of the size or length of its arguments. Return the sum of the second smallest element at odd position ‘arr’ and the second largest element at the even position.

**Assumption**

o Every array element is unique.

o Array is 0 indexed.

**Note:**

o If the array is empty, return 0.

o If array length is 3 or <3, return 0.

Example

Input:  
 Arr: 3 2 1 7 5 4

Output:  
 7

**Explanation**

o The second largest element at the even position is 3.

o The second smallest element at the odd position is 4.

o The output is 7 (3 + 4).

To solve this problem, you need to follow these steps in your C program:

1. **Handle Edge Cases**: If the array is empty or has fewer than 3 elements, return 0.
2. **Identify Elements at Odd and Even Positions**: Extract elements at odd and even indices separately.
3. **Find the Second Smallest Element at Odd Positions**: Sort the elements at odd indices and select the second smallest.
4. **Find the Second Largest Element at Even Positions**: Sort the elements at even indices and select the second largest.
5. **Sum these two elements and return the result**.

Here’s a complete C program to perform the above steps:

**#include <stdio.h>**

**#include <stdlib.h>**

**// Function to compare two integers for qsort**

**int compare(const void \*a, const void \*b) {**

**return (\*(int\*)a - \*(int\*)b);**

**}**

**// Function to find the second smallest element in a sorted array**

**int find\_second\_smallest(int arr[], int size) {**

**if (size < 2) {**

**return -1; // Not enough elements for the second smallest**

**}**

**return arr[1];**

**}**

**// Function to find the second largest element in a sorted array**

**int find\_second\_largest(int arr[], int size) {**

**if (size < 2) {**

**return -1; // Not enough elements for the second largest**

**}**

**return arr[size - 2];**

**}**

**int main() {**

**int arr[] = {3, 2, 1, 7, 5, 4};**

**int n = sizeof(arr) / sizeof(arr[0]);**

**if (n < 3) {**

**// Array has fewer than 3 elements**

**printf("0\n");**

**return 0;**

**}**

**// Arrays to store elements at odd and even positions**

**int odd\_elements[(n + 1) / 2];**

**int even\_elements[n / 2 + 1];**

**int odd\_count = 0, even\_count = 0;**

**// Separate elements into odd and even position arrays**

**for (int i = 0; i < n; i++) {**

**if (i % 2 == 0) {**

**even\_elements[even\_count++] = arr[i];**

**} else {**

**odd\_elements[odd\_count++] = arr[i];**

**}**

**}**

**// Sort odd and even elements arrays**

**qsort(odd\_elements, odd\_count, sizeof(int), compare);**

**qsort(even\_elements, even\_count, sizeof(int), compare);**

**// Find second smallest at odd positions**

**int second\_smallest\_odd = find\_second\_smallest(odd\_elements, odd\_count);**

**// Find second largest at even positions**

**int second\_largest\_even = find\_second\_largest(even\_elements, even\_count);**

**if (second\_smallest\_odd == -1 || second\_largest\_even == -1) {**

**// Not enough elements for second smallest or second largest**

**printf("0\n");**

**return 0;**

**}**

**// Calculate the result**

**int result = second\_smallest\_odd + second\_largest\_even;**

**printf("%d\n", result);**

**return 0;**

**}**

**Explanation:**

1. **Edge Cases Handling**:
   * If the array length is less than 3, output 0.
2. **Separating Odd and Even Indexed Elements**:
   * Use two arrays, odd\_elements and even\_elements, to store the elements based on their indices.
3. **Sorting**:
   * Use qsort to sort the odd and even position arrays.
4. **Finding Second Smallest and Second Largest**:
   * Use helper functions find\_second\_smallest and find\_second\_largest to get the required elements.
5. **Calculating and Printing the Result**:
   * Sum the second smallest odd-positioned element and the second largest even-positioned element, and print the result.

This program efficiently handles the given task by leveraging sorting and array manipulation techniques in C.

**######################################################################################################################################################**

**3. Write a function to validate if the provided two strings are anagrams or not. If the two strings are anagrams, then return ‘yes’. Otherwise, return ‘no’.**

Input:

Input 1: 1st string  
 Input 2: 2nd string

Output:  
 (If they are anagrams, the function will return ‘yes’. Otherwise, it will return ‘no’.)

Example

Input 1: Listen  
 Input 2: Silent

Output:  
 Yes

Explanation

Listen and Silent are anagrams (an anagram is a word formed by rearranging the letters of the other word).

To determine if two strings are anagrams, you need to verify if both strings contain the exact same characters with the exact same frequencies. Here's a concise approach to achieve this:

1. **Check Length**: If the lengths of the two strings are different, they cannot be anagrams.
2. **Sort and Compare**: One straightforward method is to sort both strings and compare them. If the sorted versions are identical, the strings are anagrams.

Below is a C function that implements this approach:

**#include <stdio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**// Function to compare two characters for qsort**

**int compare\_char(const void \*a, const void \*b) {**

**return (\*(char\*)a - \*(char\*)b);**

**}**

**// Function to check if two strings are anagrams**

**const char\* are\_anagrams(const char\* str1, const char\* str2) {**

**// Check if lengths are different**

**if (strlen(str1) != strlen(str2)) {**

**return "no";**

**}**

**// Create copies of the strings to sort**

**char \*sorted\_str1 = malloc(strlen(str1) + 1);**

**char \*sorted\_str2 = malloc(strlen(str2) + 1);**

**if (sorted\_str1 == NULL || sorted\_str2 == NULL) {**

**return "no"; // Memory allocation failed**

**}**

**// Copy the original strings**

**strcpy(sorted\_str1, str1);**

**strcpy(sorted\_str2, str2);**

**// Sort the strings**

**qsort(sorted\_str1, strlen(sorted\_str1), sizeof(char), compare\_char);**

**qsort(sorted\_str2, strlen(sorted\_str2), sizeof(char), compare\_char);**

**// Compare the sorted strings**

**int result = strcmp(sorted\_str1, sorted\_str2);**

**free(sorted\_str1);**

**free(sorted\_str2);**

**// Return the result**

**return (result == 0) ? "yes" : "no";**

**}**

**// Example usage**

**int main() {**

**const char\* str1 = "listen";**

**const char\* str2 = "silent";**

**printf("Are the strings anagrams? %s\n", are\_anagrams(str1, str2));**

**return 0;**

**}**

**Explanation:**

1. **Function are\_anagrams**:
   * **Check Length**: If the lengths of str1 and str2 differ, they cannot be anagrams.
   * **Allocate Memory**: Allocate memory for copies of the strings to sort them.
   * **Copy Strings**: Copy the original strings to avoid modifying them.
   * **Sort Strings**: Use qsort to sort both strings.
   * **Compare Sorted Strings**: Use strcmp to compare the sorted strings.
   * **Free Memory**: Free the allocated memory to prevent memory leaks.
   * **Return Result**: Return "yes" if the sorted strings match; otherwise, return "no".
2. **Memory Management**:
   * Allocate memory for the copies of the strings to sort them and free it afterward.
3. **Usage Example**:
   * The main function demonstrates how to use the are\_anagrams function.

This approach is efficient and straightforward, leveraging sorting and comparison to determine if two strings are anagrams.

**######################################################################################################################################################**

**4. Perform the following function.**

def Productsmallpair(sum,arr)

This function accepts the integers sum and arr. It is used to find the arr(j) and arr(k), where k ! = j. arr(j) and arr(k) should be the smallest elements in the array.

Keep this in mind:

o If n<2 or empty, return –1.

o If these pairs are not found, then return to zero.

o Make sure all the values are within the integer range.

Example

Input:  
 Sum: 9  
 Arr: 5 4 2 3 9 1 7

Output:  
 2

**Explanation**

From the array of integers, we have to select the two smallest numbers, i.e 2 and 1. Sum of these two (2 + 1) = 3. This is less than 9 (3 < 9). The product of these two is 2 (2 x 1 = 2) Hence the output is integer 2.

Sample input:  
 Sum: 4  
 Arr: 9 8 –7 3 9 3

Sample output:  
 –21  
 To solve the problem of finding two distinct elements in an array that add up to a given sum, you can follow these steps:

1. **Edge Case Handling**: If the array has fewer than 2 elements or is empty, return -1.
2. **Using a Hash Map**: To efficiently find pairs of elements that sum up to a given value, use a hash map (or an equivalent data structure) to keep track of the elements that have been seen.
3. **Iterate and Check**: For each element, check if there exists another element in the hash map that, together with the current element, sums up to the target value.

Here's the C program implementing this approach:

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_SIZE 1000 // Define a maximum size for the array**

**// Structure for hash map entry**

**typedef struct {**

**int key;**

**int value;**

**} HashMapEntry;**

**// Function to find the pair of elements that sum up to the target**

**int find\_pair\_with\_sum(int sum, int arr[], int n) {**

**if (n < 2) {**

**return -1; // Array has fewer than 2 elements**

**}**

**// Use a hash map to keep track of the elements**

**HashMapEntry hash\_map[MAX\_SIZE];**

**int hash\_map\_size = 0;**

**for (int i = 0; i < n; ++i) {**

**int complement = sum - arr[i];**

**// Check if the complement exists in the hash map**

**for (int j = 0; j < hash\_map\_size; ++j) {**

**if (hash\_map[j].key == complement) {**

**// We found the pair**

**printf("Found pair: (%d, %d)\n", complement, arr[i]);**

**return 1; // Return 1 to indicate that the pair was found**

**}**

**}**

**// Add the current element to the hash map**

**if (hash\_map\_size < MAX\_SIZE) {**

**hash\_map[hash\_map\_size].key = arr[i];**

**hash\_map[hash\_map\_size].value = 1;**

**++hash\_map\_size;**

**} else {**

**return 0; // Hash map is full, return 0**

**}**

**}**

**return 0; // Pair not found**

**}**

**int main() {**

**int arr[] = {1, 4, 6, 8, 10};**

**int n = sizeof(arr) / sizeof(arr[0]);**

**int sum = 14;**

**int result = find\_pair\_with\_sum(sum, arr, n);**

**if (result == -1) {**

**printf("-1\n"); // Not enough elements**

**} else if (result == 0) {**

**printf("0\n"); // Pair not found**

**}**

**return 0;**

**}**

**Explanation:**

1. **Edge Case Handling**:
   * The function find\_pair\_with\_sum checks if there are fewer than 2 elements. If true, it returns -1.
2. **Hash Map Usage**:
   * A hash map (array of HashMapEntry structures) is used to keep track of elements that have been processed.
   * For each element in the array, compute the complement (i.e., sum - arr[i]).
   * Check if the complement is already in the hash map.
   * If found, print the pair and return 1.
3. **Adding to Hash Map**:
   * If the complement is not found, add the current element to the hash map.
4. **Handling Full Hash Map**:
   * If the hash map exceeds its maximum size, the program returns 0.
5. **Main Function**:
   * Sets up an example array and target sum, then calls find\_pair\_with\_sum to find and print the result.

This approach efficiently finds pairs of elements that add up to the given sum, ensuring it works well within integer range constraints. Adjust the MAX\_SIZE based on your needs or constraints for the hash map size.

**######################################################################################################################################################**

**5. Perform the function for the given purpose.**

For writing numbers, there is a system called N-base notation. This system uses only N-based symbols. It uses symbols that are listed as the first n symbols. Decimal and n-based notations are 0:0, 1:1, 2:2, …, 10:A, 11:B, …, 35:Z.

Perform the function: Chats DectoNBase(int n, int num)

This function only uses positive integers. Use a positive integer n and num to find out the n-base that is equal to num.

**Steps**

o Select a decimal number and divide it by n. Consider this as an integer division.

o Denote the remainder as n-based notation.

o Again divide the quotient by n.

o Repeat the above steps until you get a 0 remainder.

o The remainders from last to first are the n-base values.

Assumption  
 1 < n < = 36

Example

Input:  
 N: 12  
 Num: 718

Output:  
 4BA

**Explanation**

| **num** | **Divisor** | **Quotient** | **Remainder** |
| --- | --- | --- | --- |
| 718 | 12 | 59 | 10(A) |
| 59 | 2 | 4 | 11(B) |
| 4 | 12 | 0 | 4(4) |

Sample input:  
 N: 21  
 Num: 5678

Sample output:  
 CI8  
 To convert a decimal number to an n-base number (where n is between 2 and 36), you can follow these steps:

1. **Define Symbols**: Use symbols 0 to 9 and A to Z for bases greater than 10. These symbols represent values from 0 to 35.
2. **Conversion Logic**:
   * Repeatedly divide the number by the base n and keep track of the remainders.
   * The remainders, read in reverse order, give the digits of the number in base n.

Here’s a complete C program that performs this conversion:

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_DIGITS 64 // Maximum length of the result string**

**// Function to convert a decimal number to n-base notation**

**void DectoNBase(int n, int num) {**

**// Validate the base**

**if (n < 2 || n > 36) {**

**printf("Invalid base. Base must be between 2 and 36.\n");**

**return;**

**}**

**// Handle the special case of 0**

**if (num == 0) {**

**printf("0\n");**

**return;**

**}**

**// Array to store the digits of the result**

**char result[MAX\_DIGITS];**

**int index = 0;**

**// Convert the decimal number to n-base**

**while (num > 0) {**

**int remainder = num % n;**

**if (remainder < 10) {**

**result[index++] = '0' + remainder;**

**} else {**

**result[index++] = 'A' + (remainder - 10);**

**}**

**num /= n;**

**}**

**// Print the result in reverse order**

**printf("The number in base %d is: ", n);**

**for (int i = index - 1; i >= 0; i--) {**

**putchar(result[i]);**

**}**

**printf("\n");**

**}**

**int main() {**

**int n, num;**

**// Input base and number**

**printf("Enter the base (2-36): ");**

**scanf("%d", &n);**

**printf("Enter the decimal number: ");**

**scanf("%d", &num);**

**// Convert and display the number**

**DectoNBase(n, num);**

**return 0;**

**}**

**Explanation:**

1. **Base Validation**:
   * The function first checks if the base n is within the valid range (2 to 36). If not, it prints an error message.
2. **Conversion Logic**:
   * **Special Case for 0**: Directly handle the case where num is 0, as its representation in any base is 0.
   * **Conversion Loop**: Use a while loop to repeatedly divide the number by n, storing the remainder (which represents the current least significant digit in the new base) in the result array.
   * **Character Mapping**: Convert the remainder to the appropriate character (0-9 for values 0-9, A-Z for values 10-35).
3. **Output**:
   * After the conversion loop, the digits are in reverse order, so the function prints them from the end of the result array to the beginning.
4. **Main Function**:
   * Reads the base and the number from the user and then calls DectoNBase to perform the conversion and print the result.

This approach ensures that the decimal number is correctly converted to the specified base, with support for bases up to 36.

**######################################################################################################################################################**

**6. Execute the function for the given purpose.**

When the sum of the digits exceeds a total of 9, a carry digit is added to the right-left of the digit. Execute the function: Int Numberofcarry(Integer num 1, Integer num 2)

**Assumption**

num1, num2 > = 0

Example

Input:  
 num1: 451  
 num2: 349

Output:  
 2

**Explanation**

When we add these two numbers from the right to the left, we get two carries. The value of each of the carries is 1. Hence, the output is the total of these two carries, i.e., 2.

Sample input:  
 num1: 23  
 num2: 563

Sample output:  
 0

To solve the problem of counting the number of carry operations that occur when adding two numbers digit by digit, you can follow these steps:

1. **Extract Digits**: Process the digits of both numbers starting from the least significant digit (rightmost digit).
2. **Simulate Addition**: Add corresponding digits of both numbers along with any carry from the previous addition.
3. **Count Carries**: Track when the sum of the digits exceeds 9, which results in a carry to the next digit.

Here's a complete C program to implement this approach:

**#include <stdio.h>**

**// Function to count the number of carry operations**

**int Numberofcarry(int num1, int num2) {**

**int carry = 0; // Carry for the next digit**

**int carryCount = 0; // Number of carry operations**

**// Process each digit until both numbers become 0**

**while (num1 > 0 || num2 > 0 || carry > 0) {**

**// Extract the last digit of both numbers**

**int digit1 = num1 % 10;**

**int digit2 = num2 % 10;**

**// Calculate the sum of the digits and carry**

**int sum = digit1 + digit2 + carry;**

**// Check if there is a carry**

**if (sum >= 10) {**

**carry = 1; // Set carry for the next digit**

**carryCount++; // Increment the carry count**

**} else {**

**carry = 0; // No carry needed**

**}**

**// Remove the last digit from both numbers**

**num1 /= 10;**

**num2 /= 10;**

**}**

**return carryCount;**

**}**

**int main() {**

**int num1, num2;**

**// Input the two numbers**

**printf("Enter the first number: ");**

**scanf("%d", &num1);**

**printf("Enter the second number: ");**

**scanf("%d", &num2);**

**// Get the number of carry operations**

**int result = Numberofcarry(num1, num2);**

**// Output the result**

**printf("Number of carry operations: %d\n", result);**

**return 0;**

**}**

**Explanation:**

1. **Function Numberofcarry**:
   * **Parameters**: Takes two integers, num1 and num2.
   * **Carry Initialization**: Initializes carry to 0 and carryCount to keep track of the number of carry operations.
   * **Loop Through Digits**: Uses a while loop to process each digit of num1 and num2 from right to left.
     + **Extract Digits**: Gets the last digit of both numbers using the modulo operator (%).
     + **Sum Calculation**: Computes the sum of the two digits and any existing carry.
     + **Carry Check**: Determines if the sum is 10 or more, which results in a carry to the next digit.
     + **Update Carry and Count**: Updates the carry for the next iteration and increments the carry count if necessary.
     + **Remove Last Digit**: Updates the numbers by removing the last digit using integer division (/).
2. **Main Function**:
   * **Input**: Prompts the user to enter two integers.
   * **Calculate Carries**: Calls the Numberofcarry function to compute the number of carry operations.
   * **Output**: Prints the result.

**Usage:**

Compile and run the program, then input the two numbers when prompted. The program will calculate and display the number of carry operations that occurred during the addition of the two numbers.

**######################################################################################################################################################**

**7. The given function has a string (str) and two characters, ch1 and ch2. Execute the function in such a way that str returns to its original string, and all the events in ch1 are replaced by ch2, and vice versa.**

Replacecharacter(Char str1, Char ch1, Int 1, Char ch2)

**Assumption**

This string has only alphabets that are in lower case.

Example

Input:  
 str: apples  
 ch1: a  
 ch2: p

Output:  
 paales

Explanation  
 All the ‘a’ in the string is replaced with ‘p’. And all the ‘p’s are replaced with ‘a’.

To solve the problem of replacing characters in a string, you need to swap occurrences of two specific characters, ch1 and ch2. Here's a step-by-step approach:

1. **Scan the String**: Iterate through each character in the string.
2. **Replace Characters**: For each character, check if it matches ch1 or ch2, and perform the necessary replacements.
3. **Output the Modified String**: Print the resulting string after replacements.

Here's a complete C program that implements this approach:

**#include <stdio.h>**

**#include <string.h>**

**// Function to replace occurrences of ch1 with ch2 and vice versa in the string**

**void Replacecharacter(char \*str, char ch1, char ch2) {**

**// Temporary variable to store the new string**

**char temp[strlen(str) + 1];**

**strcpy(temp, str); // Copy original string to temp**

**// Perform the character replacement**

**for (int i = 0; temp[i] != '\0'; i++) {**

**if (temp[i] == ch1) {**

**temp[i] = ch2; // Replace ch1 with ch2**

**} else if (temp[i] == ch2) {**

**temp[i] = ch1; // Replace ch2 with ch1**

**}**

**}**

**// Copy the modified string back to the original string**

**strcpy(str, temp);**

**}**

**int main() {**

**// Input variables**

**char str[100];**

**char ch1, ch2;**

**// Input the string**

**printf("Enter the string (lowercase letters only): ");**

**scanf("%99s", str); // Read string with max length 99 to avoid buffer overflow**

**// Input the characters to be swapped**

**printf("Enter the first character to replace: ");**

**scanf(" %c", &ch1); // Note the space before %c to consume any leftover newline**

**printf("Enter the second character to replace: ");**

**scanf(" %c", &ch2);**

**// Replace characters**

**Replacecharacter(str, ch1, ch2);**

**// Output the modified string**

**printf("Modified string: %s\n", str);**

**return 0;**

**}**

**Explanation:**

1. **Function Replacecharacter**:
   * **Parameters**: Takes a string str, and two characters ch1 and ch2.
   * **Temporary Storage**: Uses a temporary array temp to store the modified version of the string. This avoids modifying the string while iterating through it.
   * **Replacement Logic**: Iterates through the string and replaces occurrences of ch1 with ch2 and vice versa.
   * **Update Original String**: Copies the modified string back to the original string.
2. **Main Function**:
   * **Input**: Reads the string and characters from the user.
   * **Replacement**: Calls the Replacecharacter function to perform the replacements.
   * **Output**: Prints the resulting modified string.

**Important Notes:**

* **Buffer Size**: Ensure the buffer size for the string is large enough to handle typical input. Here, 100 characters are allocated for the string.
* **Input Handling**: The space before %c in scanf for characters ensures that any newline character left in the input buffer is ignored.

This code reads the string and characters from the user, processes the string to swap the specified characters, and prints the modified string.

**######################################################################################################################################################**

**8. Perform the function: Int operationchoices(int c, int n, int a, int b). This function considers three positive inputs of a, b and c.**

Execute the function to get:  
 (a + b), if c = 1  
 (a / b), if c = 4  
 (a – b), if c = 2  
 (a x b), if c = 3

Example:

Input:  
 a: 12  
 b: 16  
 c: 1

Output:  
 28

Explanation  
 C = 1, hence the function is (a + b). Hence, the output is 28.

Sample input:  
 a: 16  
 b: 20  
 c: 2

Sample output:  
 –4

To implement the function operationchoices that performs different operations based on the value of c, you can use a switch statement or a series of if-else conditions. The operations are:

* **Addition** (a + b) when c equals 1
* **Division** (a / b) when c equals 4
* **Subtraction** (a - b) when c equals 2
* **Multiplication** (a \* b) when c equals 3

Here's a complete C program implementing this function:

**#include <stdio.h>**

**// Function to perform the operation based on the value of c**

**int operationchoices(int c, int a, int b) {**

**int result;**

**// Perform the operation based on the value of c**

**switch (c) {**

**case 1:**

**result = a + b;**

**break;**

**case 2:**

**result = a - b;**

**break;**

**case 3:**

**result = a \* b;**

**break;**

**case 4:**

**// Check for division by zero**

**if (b != 0) {**

**result = a / b;**

**} else {**

**printf("Error: Division by zero.\n");**

**return -1; // Return an error code**

**}**

**break;**

**default:**

**printf("Error: Invalid choice of operation.\n");**

**return -1; // Return an error code**

**}**

**return result;**

**}**

**int main() {**

**int c, a, b;**

**// Input the operation choice and numbers**

**printf("Enter the operation choice (1 for addition, 2 for subtraction, 3 for multiplication, 4 for division): ");**

**scanf("%d", &c);**

**printf("Enter the first number (a): ");**

**scanf("%d", &a);**

**printf("Enter the second number (b): ");**

**scanf("%d", &b);**

**// Perform the operation and get the result**

**int result = operationchoices(c, a, b);**

**// Check if the operation was successful and print the result**

**if (result != -1) {**

**printf("Result: %d\n", result);**

**}**

**return 0;**

**}**

**Explanation:**

1. **Function operationchoices**:
   * **Parameters**: Takes three integers: c (choice of operation), a (first operand), and b (second operand).
   * **Switch Statement**: Chooses the operation based on the value of c.
     + **Addition**: When c is 1, it computes a + b.
     + **Subtraction**: When c is 2, it computes a - b.
     + **Multiplication**: When c is 3, it computes a \* b.
     + **Division**: When c is 4, it computes a / b but first checks if b is zero to avoid division by zero.
   * **Error Handling**: If c is not one of the expected values or if division by zero is attempted, it prints an error message and returns -1.
2. **Main Function**:
   * **Input**: Prompts the user to enter the choice of operation and the operands.
   * **Call Function**: Calls operationchoices to perform the calculation.
   * **Output**: Prints the result if the operation was successful; otherwise, it handles errors by printing appropriate messages.

**Important Notes:**

* **Division by Zero**: The function includes a check to ensure that division by zero is handled gracefully.
* **Error Codes**: Returns -1 in case of invalid operation choices or errors.

This program will correctly perform the specified operations based on user input and handle possible errors gracefully.

**######################################################################################################################################################**

**9. Perform the function Int calculate(int m, int n). This function needs two positive integers. Calculate the sum of numbers between these two numbers that are divisible by 3 and 5.**

Assumption  
 m > n > = 0

Example

Input:  
 m: 12  
 n: 50

Output:  
 90

Explanation  
 The numbers between 12 and 50 that are divisible by 3 and 5 are 15, 30, and 45. The sum of these numbers is 90.

Sample input:  
 m: 100  
 n: 160

Sample output:  
 405

To calculate the sum of numbers between two positive integers m and n (where m is greater than or equal to n) that are divisible by both 3 and 5, you can follow these steps:

1. **Initialize a Sum Variable**: Keep track of the total sum of the numbers that meet the criteria.
2. **Loop Through the Range**: Iterate from n to m, and for each number in this range, check if it is divisible by both 3 and 5.
3. **Update the Sum**: If a number is divisible by both 3 and 5, add it to the sum.

Here's a complete C program that performs these steps:

**#include <stdio.h>**

**// Function to calculate the sum of numbers between m and n that are divisible by both 3 and 5**

**int calculate(int m, int n) {**

**int sum = 0;**

**// Iterate from n to m**

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

**// Check if the number is divisible by both 3 and 5**

**if (i % 3 == 0 && i % 5 == 0) {**

**sum += i;**

**}**

**}**

**return sum;**

**}**

**int main() {**

**int m, n;**

**// Input the values of m and n**

**printf("Enter the value of m (m >= n): ");**

**scanf("%d", &m);**

**printf("Enter the value of n (n >= 0): ");**

**scanf("%d", &n);**

**// Check for valid input**

**if (m < n || n < 0) {**

**printf("Invalid input. Ensure that m >= n and n >= 0.\n");**

**return 1; // Return with an error code**

**}**

**// Calculate the result**

**int result = calculate(m, n);**

**// Output the result**

**printf("The sum of numbers between %d and %d that are divisible by both 3 and 5 is: %d\n", m, n, result);**

**return 0;**

**}**

**Explanation:**

1. **Function calculate**:
   * **Parameters**: Takes two integers, m and n, where m is greater than or equal to n.
   * **Sum Initialization**: Initializes sum to 0.
   * **Loop Through Range**: Iterates from n to m (inclusive).
     + **Divisibility Check**: Checks if the current number i is divisible by both 3 and 5 using the modulo operator (%).
     + **Update Sum**: If the number meets the criteria, it is added to sum.
2. **Main Function**:
   * **Input**: Prompts the user to enter values for m and n.
   * **Validation**: Checks if m is greater than or equal to n and if n is non-negative.
   * **Calculation**: Calls the calculate function to get the sum of numbers divisible by both 3 and 5.
   * **Output**: Prints the result.

**Important Notes:**

* **Input Validation**: The program checks if the input values are valid before proceeding with the calculation.
* **Range Iteration**: The loop iterates through the entire range from n to m, ensuring that all potential numbers are considered.

This program will compute the desired sum based on the provided input values and handle potential edge cases such as invalid input.

**######################################################################################################################################################**

**10. Execute the function for the given purpose.**

Create a matrix and mention the elements in it. Now, divide the main matrix into two halves in such a way that the elements in index 0 are even, the elements in index 1 are odd, and so on.

Then arrange the values in ascending order for even and odd. After this, calculate the sum of the second largest numbers from both even and odd matrices.

Example  
 The size of the array is 5.  
 Element at 0 index: 3  
 Element at 1 index: 4  
 Element at 2 index: 1  
 Element at 3 index: 7  
 Element at 4 index: 9

Even array: 1,3,9  
 Odd array: 4,7

To achieve the task described, you need to follow these steps:

1. **Create the Matrix**: Store the matrix elements based on user input.
2. **Divide into Even and Odd Arrays**: Separate the elements based on their index (even-indexed and odd-indexed).
3. **Sort the Arrays**: Arrange the values in ascending order for both even and odd arrays.
4. **Find the Second Largest Number**: Calculate the sum of the second largest numbers from both sorted arrays.

Here’s a complete C program that performs these steps:

**#include <stdio.h>**

**#include <stdlib.h>**

**// Function to compare two integers for qsort**

**int compare(const void \*a, const void \*b) {**

**return (\*(int \*)a - \*(int \*)b);**

**}**

**// Function to find the second largest number in an array**

**int secondLargest(int arr[], int size) {**

**if (size < 2) {**

**return -1; // Not enough elements to find second largest**

**}**

**int largest = arr[size - 1];**

**int secondLargest = -1;**

**// Find the second largest element**

**for (int i = size - 2; i >= 0; i--) {**

**if (arr[i] < largest) {**

**secondLargest = arr[i];**

**break;**

**}**

**}**

**return secondLargest;**

**}**

**// Function to process the matrix and calculate the required sum**

**int processMatrix(int matrix[], int size) {**

**// Arrays to hold even and odd indexed elements**

**int evenArray[size / 2 + 1]; // +1 for odd-sized arrays**

**int oddArray[size / 2 + 1]; // +1 for odd-sized arrays**

**int evenCount = 0;**

**int oddCount = 0;**

**// Separate elements into even and odd arrays based on their index**

**for (int i = 0; i < size; i++) {**

**if (i % 2 == 0) {**

**evenArray[evenCount++] = matrix[i];**

**} else {**

**oddArray[oddCount++] = matrix[i];**

**}**

**}**

**// Sort the even and odd arrays**

**qsort(evenArray, evenCount, sizeof(int), compare);**

**qsort(oddArray, oddCount, sizeof(int), compare);**

**// Find the second largest in both sorted arrays**

**int secondLargestEven = secondLargest(evenArray, evenCount);**

**int secondLargestOdd = secondLargest(oddArray, oddCount);**

**// Calculate and return the sum of second largest numbers**

**return (secondLargestEven != -1 ? secondLargestEven : 0) +**

**(secondLargestOdd != -1 ? secondLargestOdd : 0);**

**}**

**int main() {**

**int size;**

**// Input the size of the array**

**printf("Enter the size of the array: ");**

**scanf("%d", &size);**

**// Validate the size**

**if (size <= 0) {**

**printf("Invalid size.\n");**

**return 1;**

**}**

**int matrix[size];**

**// Input elements of the array**

**for (int i = 0; i < size; i++) {**

**printf("Element at index %d: ", i);**

**scanf("%d", &matrix[i]);**

**}**

**// Process the matrix and calculate the result**

**int result = processMatrix(matrix, size);**

**// Output the result**

**printf("The sum of the second largest numbers from both arrays is: %d\n", result);**

**return 0;**

**}**

**Explanation:**

1. **Function compare**:
   * A comparison function used by qsort to sort integers in ascending order.
2. **Function secondLargest**:
   * Finds the second largest number in an array. It returns -1 if there are not enough elements.
3. **Function processMatrix**:
   * **Initialization**: Creates arrays for even and odd indexed elements. Uses size / 2 + 1 to handle cases where the number of elements is odd.
   * **Separation**: Iterates through the input matrix, populating even and odd arrays based on index parity.
   * **Sorting**: Uses qsort to sort both arrays.
   * **Second Largest Calculation**: Finds the second largest element in each sorted array and calculates their sum.
4. **Main Function**:
   * **Input**: Reads the size of the array and its elements from the user.
   * **Validation**: Ensures the size is positive.
   * **Processing**: Calls processMatrix to compute the sum of second largest elements.
   * **Output**: Prints the final result.

This program correctly separates, sorts, and computes the required values based on the given input, ensuring accurate results for various edge cases.

**######################################################################################################################################################**

**11. The binary number system only uses two digits 1 and 0.**

Perform the function: Int OperationsBinarystring(char\* str)

**Assumptions**

o Return to –1 if str is null.

o The str is odd.

Example:

Input:  
 Str: ICOCICIAOBI

Output:  
 1

Explanation  
 The input when expanded is “1 XOR 0 XOR 1 XOR 1 XOR 1 AND 0 OR 1”. The result becomes 1 and hence the output is 1.

To solve the problem of evaluating a binary string expression that includes XOR, AND, and OR operations, follow these steps:

1. **Parse the String**: Convert the given string into a sequence of binary operations.
2. **Evaluate the Expression**: Perform the operations according to their precedence (AND, then OR, then XOR) on binary values.

**Steps to Implement:**

1. **Check for Null Input**: Return -1 if the input string is null.
2. **Parse and Convert**: Convert the given string into a series of binary operations and values.
3. **Evaluate**: Perform the binary operations in the correct order.

Here’s a C program to achieve this:

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**// Function to evaluate binary operations**

**int OperationsBinarystring(char\* str) {**

**if (str == NULL) {**

**return -1; // Null input**

**}**

**// Check if the length of the string is odd**

**int len = strlen(str);**

**if (len % 2 == 0) {**

**return -1; // Invalid length**

**}**

**// Process the string**

**int result = str[0] == '1' ? 1 : 0; // Initialize result with the first character**

**// Iterate through the string, processing operations**

**for (int i = 1; i < len; i += 2) {**

**char op = str[i]; // Operation (XOR, AND, OR)**

**int value = str[i + 1] == '1' ? 1 : 0; // Binary value (1 or 0)**

**switch (op) {**

**case 'X':**

**result ^= value; // XOR operation**

**break;**

**case 'A':**

**result &= value; // AND operation**

**break;**

**case 'O':**

**result |= value; // OR operation**

**break;**

**default:**

**return -1; // Invalid operation**

**}**

**}**

**return result;**

**}**

**int main() {**

**char str[100];**

**// Input the binary string**

**printf("Enter the binary string (odd length, with only '1', '0', 'X', 'A', 'O'): ");**

**scanf("%99s", str);**

**// Evaluate the binary string**

**int result = OperationsBinarystring(str);**

**// Output the result**

**if (result == -1) {**

**printf("Error: Invalid input.\n");**

**} else {**

**printf("Output: %d\n", result);**

**}**

**return 0;**

**}**

**Explanation:**

1. **Function OperationsBinarystring**:
   * **Null Check**: If the input string is null, return -1.
   * **Length Check**: Ensure that the length of the string is odd.
   * **Initialization**: Set the initial value based on the first character (1 or 0).
   * **Processing**:
     + **Loop**: Iterates through the string, processing each operation (X, A, O for XOR, AND, OR) on the current value.
     + **Operations**:
       - **XOR (^)**: Exclusive OR.
       - **AND (&)**: Logical AND.
       - **OR (|)**: Logical OR.
   * **Default Case**: Handles invalid operations.
2. **Main Function**:
   * **Input**: Reads the binary string from the user.
   * **Evaluation**: Calls OperationsBinarystring to get the result.
   * **Output**: Prints the result or an error message if the input is invalid.

**Notes:**

* The string should only contain binary digits (1 and 0) and the operations (X, A, O).
* The program assumes that operations are always correctly formatted and separated by binary digits.

This program efficiently parses and evaluates binary operations from a given string, providing the correct result or error messages as needed.

**######################################################################################################################################################**

**12. Perform the function Checkpassword (char str[], int n)**

Execute the function which happens to be 1 if the str is a valid password or it remains 0.

Conditions for a valid password:

o The password should have at least 4 characters.

o It should have at least 1 digit.

o It should have one capital letter.

o It should not have any spaces or obliques (/).

o The first character should not be a number.

Assumption  
 The input is not empty.

Example

Input:  
 aA1\_67

Output:  
 1  
 To implement the function Checkpassword that validates a password based on the given conditions, follow these steps:

1. **Check Length**: Ensure the password is at least 4 characters long.
2. **Check for Digits**: Ensure the password contains at least one digit.
3. **Check for Uppercase Letters**: Ensure the password contains at least one uppercase letter.
4. **Check for Invalid Characters**: Ensure the password does not contain spaces or slashes (/).
5. **Check Starting Character**: Ensure the first character is not a digit.

Here's a C program to perform the password validation:

**#include <stdio.h>**

**#include <ctype.h> // For isdigit() and isupper()**

**// Function to check if the password is valid**

**int Checkpassword(char str[], int n) {**

**// Check if the password length is at least 4 characters**

**if (n < 4) {**

**return 0;**

**}**

**// Variables to check for digits, uppercase letters, and invalid characters**

**int hasDigit = 0;**

**int hasUpper = 0;**

**int hasInvalidChar = 0;**

**// Check if the first character is a digit**

**if (isdigit(str[0])) {**

**return 0;**

**}**

**// Iterate through the password characters**

**for (int i = 0; i < n; i++) {**

**if (isdigit(str[i])) {**

**hasDigit = 1;**

**} else if (isupper(str[i])) {**

**hasUpper = 1;**

**} else if (str[i] == ' ' || str[i] == '/') {**

**hasInvalidChar = 1;**

**break;**

**}**

**}**

**// Validate the password based on the criteria**

**if (hasDigit && hasUpper && !hasInvalidChar) {**

**return 1;**

**} else {**

**return 0;**

**}**

**}**

**int main() {**

**char str[100];**

**int n;**

**// Input the password**

**printf("Enter the password: ");**

**fgets(str, sizeof(str), stdin);**

**// Remove trailing newline character if present**

**str[strcspn(str, "\n")] = 0;**

**// Get the length of the password**

**n = strlen(str);**

**// Check if the password is valid**

**int result = Checkpassword(str, n);**

**// Output the result**

**printf("%d\n", result);**

**return 0;**

**}**

**Explanation:**

1. **Function Checkpassword**:
   * **Length Check**: Returns 0 if the password length is less than 4 characters.
   * **First Character Check**: Returns 0 if the first character is a digit.
   * **Character Analysis**: Iterates through the string to check for the presence of digits and uppercase letters, and invalid characters (spaces or slashes).
   * **Validation**: Returns 1 if the password meets all the criteria (contains at least one digit, one uppercase letter, no invalid characters, and starts with a non-digit character); otherwise, returns 0.
2. **Main Function**:
   * **Input**: Reads the password from the user.
   * **Trim Newline**: Removes the trailing newline character added by fgets.
   * **Length Calculation**: Calculates the length of the input string.
   * **Validation**: Calls Checkpassword and prints the result.

**Important Notes:**

* **Character Checks**: Uses isdigit() to check for digits and isupper() to check for uppercase letters.
* **Invalid Characters**: Checks explicitly for spaces and slashes.
* **String Handling**: Ensures proper input handling and validation.

This program efficiently checks for password validity according to the specified criteria and provides appropriate feedback based on the input.

**############################################################################################################################################################**

**13. Execute this function Void MaxInArray(int arr[], int length)**

This function helps in finding the maximum element in the array. Execute this function to print the maximum element in the array with its index.

**Assumptions**

o The index in the array for all the elements starts at 0.

o The maximum element is in a different line in the output.

o There has to be only one maximum element.

o The function prints only what is required.

Example

Input:  
 23 45 82 27 66 12 78 13 71 86

Output:  
 86  
 9

Explanation  
 The maximum element is 86 and the array is 9.  
 o create a C program that includes the MaxInArray function, which finds the maximum element in an array and its index, follow these steps:

1. **Define the MaxInArray function**: This function will iterate through the array to find the maximum element and its index.
2. **Implement the main function**: This function will read input values into an array, call the MaxInArray function, and print the results.

Here's a complete C program that performs the required tasks:

**#include <stdio.h>**

**void MaxInArray(int arr[], int length) {**

**if (length <= 0) {**

**return; // handle case of empty array**

**}**

**int max = arr[0];**

**int maxIndex = 0;**

**for (int i = 1; i < length; i++) {**

**if (arr[i] > max) {**

**max = arr[i];**

**maxIndex = i;**

**}**

**}**

**printf("%d\n", max);**

**printf("%d\n", maxIndex);**

**}**

**int main() {**

**int arr[3]; // assuming a maximum of 100 elements**

**int length = 0;**

**// Read integers from input**

**while ( length<3) {**

**scanf("%d", &arr[length]);**

**length++;**

**}**

**// Call the function**

**MaxInArray(arr, length);**

**return 0;**

**}**

### **Explanation:**

1. **Function MaxInArray**:
   * **Parameters**: arr[] (array of integers), length (number of elements in the array).
   * It first checks if the array length is greater than zero.
   * It initializes max with the first element of the array and maxIndex to 0.
   * It then iterates through the array. If a larger element is found, it updates max and maxIndex.
   * Finally, it prints the maximum element and its index on separate lines.
2. **Function main**:
   * It defines an array arr with a size of 100 (assuming the input won't exceed 100 elements).
   * It reads integers from the standard input until EOF is encountered (using scanf in a loop).
   * It then calls MaxInArray to process the input array and print the results.

**############################################################################################################################################################**

**14. Change frequent character**

The function to execute is  
 Chat*Frequentcharacter(Char* str, char x)

This function has a string and a character. This function requires replacing the most used character in the str with the ‘x’ character.

Note: If two characters have the same frequency, then we will have to consider the frequency which has the lower ASCII value.

Example

Input:  
 str: bbadbbababb  
 char x: t

Output:  
 ttadttatatt

Explanation  
 The maximum character repeated is ‘b’ that is replaced with ‘t’.

To solve this problem, you'll need to write a C program that includes a function ChatFrequentcharacter. This function will replace the most frequently occurring character in the string with a specified character. If there are multiple characters with the same frequency, the one with the lower ASCII value should be replaced.

Here's a complete C program to achieve this:

**#include <stdio.h>**

**#include <string.h>**

**#define MAX\_CHAR 256 // Number of possible ASCII characters**

**void ChatFrequentcharacter(char str[], char x) {**

**int freq[MAX\_CHAR] = {0}; // Frequency array to count occurrences of each character**

**int i;**

**int maxFreq = 0;**

**char maxChar = '\0';**

**// Calculate frequency of each character**

**for (i = 0; str[i] != '\0'; i++) {**

**freq[(unsigned char)str[i]]++;**

**}**

**// Find the character with the maximum frequency**

**for (i = 0; i < MAX\_CHAR; i++) {**

**if (freq[i] > maxFreq || (freq[i] == maxFreq && i < (unsigned char)maxChar)) {**

**maxFreq = freq[i];**

**maxChar = (char)i;**

**}**

**}**

**// Replace all occurrences of maxChar with x**

**for (i = 0; str[i] != '\0'; i++) {**

**if (str[i] == maxChar) {**

**str[i] = x;**

**}**

**}**

**// Print the modified string**

**printf("%s\n", str);**

**}**

**int main() {**

**char str[100];**

**char x;**

**// Read input**

**printf("Enter the string: ");**

**fgets(str, sizeof(str), stdin);**

**// Remove trailing newline character from fgets**

**str[strcspn(str, "\n")] = '\0';**

**printf("Enter the replacement character: ");**

**scanf(" %c", &x);**

**// Call the function**

**ChatFrequentcharacter(str, x);**

**return 0;**

**}**

### **Explanation:**

1. **Function ChatFrequentcharacter**:
   * **Parameters**: str (the input string), x (the replacement character).
   * **Frequency Calculation**: It uses an array freq to store the frequency of each ASCII character. The size of this array is 256 to cover all possible ASCII characters.
   * **Finding the Most Frequent Character**: It iterates over the freq array to find the character with the highest frequency. In case of a tie (same frequency), the character with the lower ASCII value is chosen.
   * **Replacement**: It replaces all occurrences of the most frequent character with the specified character x.
   * **Output**: It prints the modified string.
2. **Function main**:
   * **Input Handling**: It reads the input string using fgets and the replacement character using scanf. Note that fgets is used to handle spaces and newlines in the input string.
   * **Function Call**: It calls ChatFrequentcharacter with the provided input.

**############################################################################################################################################################**

**15. Execute the function Def Autocount(n).**

The function accepts the string n. It checks whether the number is an autobiographical number or not. If an integer returns, then it is an autobiographical number. If 0 returns, then it is not an autobiographical number.

**Assumption**

o The input value should not be more than 10 characters.

o The input string will have numeric characters.

Example

Input:  
 N: “1210”

Output:  
 3

Explanation  
 The 0th position has the number 1, the 1st position has the number 2, the 2nd position has the number 1, and the 3rd position has number 0. Hence, it is an autobiographical number.

The count of autobiographical numbers in the input is 3, hence 3 is returned.

An autobiographical number is a number where the digit at each position indicates the count of digits from 0 to 9 that appear in the number. For example, in the number 1210, the digit at each position indicates:

* 1 zeroes
* 2 ones
* 1 twos
* 0 threes

Thus, the number 1210 is autobiographical.

Here's a C program that includes a function DefAutocount which checks whether the given string represents an autobiographical number:

**#include <stdio.h>**

**#include <string.h>**

**#include <ctype.h>**

**#define MAX\_LENGTH 10**

**int DefAutocount(char n[]) {**

**int length = strlen(n);**

**int count[10] = {0};**

**// Validate input length**

**if (length > MAX\_LENGTH) {**

**return 0;**

**}**

**// Validate if all characters are digits**

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

**if (!isdigit(n[i])) {**

**return 0;**

**}**

**}**

**// Count occurrences of each digit**

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

**count[n[i] - '0']++;**

**}**

**// Check if the number is autobiographical**

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

**if (count[i] != (n[i] - '0')) {**

**return 0;**

**}**

**}**

**return 1;**

**}**

**int main() {**

**char n[MAX\_LENGTH + 1];**

**// Read input**

**printf("Enter the number (max 10 digits): ");**

**scanf("%10s", n); // Limit input to 10 characters**

**// Call the function and print the result**

**int result = DefAutocount(n);**

**if (result) {**

**printf("%d\n", strlen(n) - 1); // Output the length of the number minus one**

**} else {**

**printf("0\n");**

**}**

**return 0;**

**}**

### **Explanation:**

1. **Function DefAutocount**:
   * **Parameters**: n[] (string representing the number).
   * **Check Length**: It ensures the length of the string does not exceed 10 characters.
   * **Validate Digits**: It verifies that all characters in the string are numeric digits.
   * **Count Digits**: It counts occurrences of each digit (0 through 9) in the string.
   * **Check Autobiographical**: It checks if the number at each position matches the count of that digit.
   * **Return Value**: Returns 1 if the number is autobiographical, otherwise returns 0.
2. **Function main**:
   * **Input Handling**: It reads up to 10 characters of numeric input.
   * **Function Call**: It calls DefAutocount and prints strlen(n) - 1 if the number is autobiographical, otherwise prints 0.

**############################################################################################################################################################**

**16. In a given list of integers, find the number that has the highest frequency. If there are one or more such numbers, give the smaller one as output.**

Input:  
 3  
 7  
 2 4 5 2 3 2 4  
 6  
 1 2 1 1 2 1  
 10  
 1 1 1 1 1 1 1 1 1 1

Output:  
 2  
 1  
 1  
 To solve this problem, we need to develop a C program that will read a list of integers and find the number with the highest frequency. If multiple numbers have the same highest frequency, we should return the smallest one.

Here's a complete C program to accomplish this:

**#include <stdio.h>**

**#include <limits.h>**

**#define MAX\_SIZE 1000 // Define a maximum size for the input array**

**int findHighestFrequencyNumber(int arr[], int length) {**

**int frequency[MAX\_SIZE] = {0}; // Frequency array**

**int maxFrequency = 0;**

**int result = INT\_MAX;**

**// Calculate frequencies of each number**

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

**frequency[arr[i]]++;**

**if (frequency[arr[i]] > maxFrequency || (frequency[arr[i]] == maxFrequency && arr[i] < result)) {**

**maxFrequency = frequency[arr[i]];**

**result = arr[i];**

**}**

**}**

**return result;**

**}**

**int main() {**

**int n;**

**int arr[MAX\_SIZE];**

**int i,z=0;**

**while ( z<3) {**

**scanf("%d", &n);**

**printf("n= %d\n",n);**

**printf("------------------------\n");**

**for (i = 0; i < n; i++) {**

**scanf("%d", &arr[i]);**

**}**

**z++;**

**int result = findHighestFrequencyNumber(arr, n);**

**printf("%d\n", result);**

**printf("------------------------\n\n");**

**}**

**return 0;**

**}**

### **Explanation:**

1. **Function findHighestFrequencyNumber**:
   * **Parameters**: arr[] (array of integers), length (number of elements in the array).
   * **Frequency Array**: The frequency array keeps track of how many times each number appears. This array assumes that the numbers are non-negative and less than MAX\_SIZE. Adjust MAX\_SIZE if needed for your specific use case.
   * **Track Maximum Frequency**: As we populate the frequency array, we also track the maximum frequency and the smallest number with that frequency.
   * **Return Value**: It returns the number with the highest frequency. In case of a tie, the smallest number is returned.
2. **Function main**:
   * **Input Handling**: Reads the number of integers (n), then reads n integers into the array arr.
   * **Function Call**: Calls findHighestFrequencyNumber to get the number with the highest frequency and prints it.
   * **Loop Until EOF**: The while loop reads multiple test cases from standard input until the end of the file.

**############################################################################################################################################################**

**17. Execute the function for the given purpose.**

Write a function mergeArrays which merges two sorted arrays to create one single sorted array. Complete the function int\* mergeArrays(int a[], int b[], int asize, int bsize) below which takes the pointers to the first element of the two sorted arrays and the size of the arrays, and returns the base address of the final sorted array.

Input:  
 4 // Size of 1st array  
 1 2 3 6 // Elements of 1st array  
 3 // Size of 2nd array  
 4 5 7 // Elements of 2nd array

Output:  
 1  
 2  
 3  
 4  
 5  
 6  
 7  
 To merge two sorted arrays into a single sorted array in C, you need to implement the mergeArrays function, which will take two sorted arrays and their sizes as input and return a pointer to the merged sorted array. Here's a complete C program that includes this function and demonstrates its use:

**#include <stdio.h>**

**#include <stdlib.h>**

**int\* mergeArrays(int a[], int b[], int asize, int bsize) {**

**int\* merged = (int\*)malloc((asize + bsize) \* sizeof(int)); // Allocate memory for the merged array**

**if (merged == NULL) {**

**fprintf(stderr, "Memory allocation failed\n");**

**exit(1);**

**}**

**int i = 0, j = 0, k = 0;**

**// Merge the two sorted arrays**

**while (i < asize && j < bsize) {**

**if (a[i] <= b[j]) {**

**merged[k++] = a[i++];**

**} else {**

**merged[k++] = b[j++];**

**}**

**}**

**// Copy the remaining elements of the first array, if any**

**while (i < asize) {**

**merged[k++] = a[i++];**

**}**

**// Copy the remaining elements of the second array, if any**

**while (j < bsize) {**

**merged[k++] = b[j++];**

**}**

**return merged;**

**}**

**int main() {**

**int asize, bsize;**

**// Read size and elements of the first array**

**printf("Enter size of 1st array: ");**

**scanf("%d", &asize);**

**int\* a = (int\*)malloc(asize \* sizeof(int));**

**if (a == NULL) {**

**fprintf(stderr, "Memory allocation failed\n");**

**return 1;**

**}**

**printf("Enter elements of 1st array: ");**

**for (int i = 0; i < asize; i++) {**

**scanf("%d", &a[i]);**

**}**

**// Read size and elements of the second array**

**printf("Enter size of 2nd array: ");**

**scanf("%d", &bsize);**

**int\* b = (int\*)malloc(bsize \* sizeof(int));**

**if (b == NULL) {**

**fprintf(stderr, "Memory allocation failed\n");**

**free(a);**

**return 1;**

**}**

**printf("Enter elements of 2nd array: ");**

**for (int i = 0; i < bsize; i++) {**

**scanf("%d", &b[i]);**

**}**

**// Merge arrays**

**int\* merged = mergeArrays(a, b, asize, bsize);**

**// Print the merged array**

**printf("Merged array:\n");**

**for (int i = 0; i < asize + bsize; i++) {**

**printf("%d\n", merged[i]);**

**}**

**// Free allocated memory**

**free(a);**

**free(b);**

**free(merged);**

**return 0;**

**}**

### **Explanation:**

1. **Function mergeArrays**:
   * **Parameters**:
     + a[]: Pointer to the first sorted array.
     + b[]: Pointer to the second sorted array.
     + asize: Size of the first array.
     + bsize: Size of the second array.
   * **Memory Allocation**: Allocates memory for the merged array which will be of size asize + bsize.
   * **Merging**: Uses two indices to traverse both arrays and merge them into the merged array.
   * **Copy Remaining Elements**: After one of the arrays is exhausted, copies the remaining elements of the other array.
   * **Return**: Returns a pointer to the merged array.
2. **Function main**:
   * **Input Handling**: Reads the sizes and elements of the two arrays.
   * **Function Call**: Calls mergeArrays to get the merged array.
   * **Output**: Prints the elements of the merged array.
   * **Memory Management**: Frees the allocated memory for the arrays.

**############################################################################################################################################################**

**18. Create web access management to the kth largest number. It will accept an integer k and an array arr as its conditions and has to return the greatest element based on the value of k. That is, if k = 0, return the greatest element. If k = 1, return the second greatest element, and so on.**

**Example**

Suppose the array contains values like {74, 85, 102, 99, 101, 56, 84} and the integer k is 2. The method will return 99, the third greatest element, as there are only two (according to the value of k) values greater than 99 (101 and 102).

To solve the problem of finding the k-th largest element in an array, you need to implement a function that sorts the array in descending order and then retrieves the element at the k-th position. Here’s a C program that accomplishes this:

**#include <stdio.h>**

**#include <stdlib.h>**

**// Function to compare two integers for qsort**

**int compare(const void \*a, const void \*b) {**

**return (\*(int\*)b - \*(int\*)a); // For descending order**

**}**

**// Function to find the k-th largest element**

**int kthLargest(int arr[], int size, int k) {**

**if (k < 0 || k >= size) {**

**fprintf(stderr, "Invalid value of k\n");**

**exit(1);**

**}**

**// Sort the array in descending order**

**qsort(arr, size, sizeof(int), compare);**

**// Return the k-th largest element**

**return arr[k];**

**}**

**int main() {**

**int size, k;**

**// Read size of the array**

**printf("Enter the size of the array: ");**

**scanf("%d", &size);**

**if (size <= 0) {**

**fprintf(stderr, "Array size must be positive\n");**

**return 1;**

**}**

**// Allocate memory for the array**

**int \*arr = (int\*)malloc(size \* sizeof(int));**

**if (arr == NULL) {**

**fprintf(stderr, "Memory allocation failed\n");**

**return 1;**

**}**

**// Read elements of the array**

**printf("Enter the elements of the array:\n");**

**for (int i = 0; i < size; i++) {**

**scanf("%d", &arr[i]);**

**}**

**// Read value of k**

**printf("Enter the value of k: ");**

**scanf("%d", &k);**

**// Find and print the k-th largest element**

**int result = kthLargest(arr, size, k);**

**printf("The %d-th largest element is: %d\n", k + 1, result);**

**// Free allocated memory**

**free(arr);**

**return 0;**

**}**

### **Explanation:**

1. **Function compare**:
   * **Purpose**: Used by qsort to sort the array in descending order.
   * **Parameters**: Takes pointers to two integers and returns the difference in reversed order to sort in descending.
2. **Function kthLargest**:
   * **Parameters**:
     + arr[]: Array of integers.
     + size: Number of elements in the array.
     + k: The index for the k-th largest element (0-based index).
   * **Sorting**: Uses qsort to sort the array in descending order.
   * **Validation**: Checks if k is within the valid range.
   * **Return**: Returns the k-th largest element.
3. **Function main**:
   * **Input Handling**: Reads the size of the array, elements of the array, and the value of k.
   * **Function Call**: Calls kthLargest to get the k-th largest element and prints it.
   * **Memory Management**: Allocates and frees memory for the array.

**############################################################################################################################################################**

**19. We have mentioned a list of integers that have no duplicates. Find how many swaps it will take to sort the list in ascending order using Bubble sort.**

Input:  
 3  
 5  
 2 1 4 6 3  
 10  
 123 21 34 45 25 675 23 44 55 900  
 1  
 23

Output:  
 3  
 16  
 0

**############################################################################################################################################################**

**20. Write a program to count the number of swaps required to sort a given list of integers in ascending order using the selection sort algorithm.**

Input:  
 2  
 3  
 4 2 5  
 5  
 10 11 8 7 1

Output:  
 1  
 3

**############################################################################################################################################################**

**21. Form an array of 1000 integers and find out the second-largest number. If there is no second largest number, return the value to –1.**

Example  
 Input 1:  
 3  
 Input 2:  
 {2,1,2}  
 Output:  
 1  
 Explanation  
 The integer 1 is the second largest in the array.  
 Example  
 Input 1:  
 5  
 Input 2:  
 {4,7,9,8,0}  
 Output:  
 8

**############################################################################################################################################################**

**22. Adam decides to do some charity work. From day 1 till day n, he will give i^2 coins to charity. On day ‘i’ (1 < = i < = n), find the number of coins he gives to charity.**

Example 1  
 Input:  
 2  
 Output:  
 5  
 Explanation:  
 There are 2 days.  
 Example 2

Input:  
 3

Output:  
 14

**############################################################################################################################################################**

**23. Perform a function to reverse a string word-wise. The input here will be the string. In the output, the last word mentioned should come as the first word and vice versa.**

Example  
 Input:  
 Welcome to code  
 Output:  
 code to Welcome

Explanation  
 The Reversed string word wise function is applied.  
 Example  
 Input:  
 Code to Crack Puzzle  
 Output:  
 Puzzle Crack to Code

**############################################################################################################################################################**

**24. Find the sum of the divisors for the N integer number.**

Example 1  
 Input:  
 6  
 Output:  
 12

Explanation  
 Divisors of 6 are 1, 2, 3, and 6. The sum of these numbers is 12.

Example 2  
 Input:  
 36  
 Output:  
 91

**############################################################################################################################################################**

**25. Execute a function that accepts the integer array of length ‘size’ and finds out the maximum number that can be formed by permutation.**

Note: You will have to rearrange the numbers to make the maximum number.

Example  
 Input:  
 34 79 58 64  
 Output:  
 98765443

Explanation  
 All digits of the array are 3, 4, 7, 9, 5, 8, 6, and 4. The maximum number found after rearranging all the digits is 98765443.

**############################################################################################################################################################**

**26. Find a string of a length of 1000 for a large number. Output is the modulo of 11. The output specification is to return the remainder modulo 11 of the input.**

Input:  
 121  
 Output:  
 0

Explanation  
 121 mod 11 = 0

**############################################################################################################################################################**

**27. Find out if the given set of points are on a straight line or not. If the points are on a straight line, then return the equation. If not, then return 0.**

Example  
 Input:  
 3  
 1 1 2 2 3 3  
 Output:  
 1x – 1y = 0

Explanation  
 The three points here are [1,1], [2,2], and [3,3]. These lie on a line, so the function returned the equation.

**############################################################################################################################################################**

**28. Write a function to find roots of a quadratic equation ax^2 + bx + c = 0.**

Note: The formula to find the roots of a quadratic equation is given below:

Example  
 Input 1: 1  
 Input 2: –2  
 Input 3: 3  
 Output:  
 {3.0,–1.0}

Explanation:  
 On substituting the values of a, b, and c in the formula, the roots will be as follows:  
 +X = 3.0  
 -X = –1.0

**############################################################################################################################################################**

**29. Write a function to find if the given string is a palindrome or not. Return 1 if the input string is a palindrome, else return 0.**

Input:  
 level  
 Output:  
 1

Explanation:  
 The reverse of the string ‘level’ is ‘level’. As they are the same, the string is a palindrome.

**############################################################################################################################################################**

**30. Write a function to check if the given two strings are anagrams or not. Return ‘Yes’ if they are anagrams, otherwise, return ‘No’.**

Example  
 Input 1: build  
 Input 2: dubli  
 Output:  
 Yes