END-SEMESTER PROJECT REPORT

Project Name	Algorithm	**Reporting Period**
Professor	Pr.Hantout	Jan 29, 2025
Name	Louafi Sohaib Mechir Islam	
section & Group	section B Group B4	

STATUS UPDATES

Task or Deliverable	Status
Operations on numbers	DONE
Operations on strings	DONE
Operations on arrays	DONE
Operations on matrices	DONE
Optional parts	DONE
Report	DONE

Report

Operations on numbers

Basic operations on numbers :

1-Sum of Digits :

Objective :

A program determines the total digit count of any provided integer.

Steps:

1. Extract each digit by using num % 10.
2. Add the digit to the sum.
3. Remove the digit from the number with num /= 10.
4. Repeat until the number is 0, then return the sum.

Example:

For num = 1234:

- Sum of digits = 1 + 2 + 3 + 4 = 10

2-Reverse Number:

Objective :

A given integer undergoes digit reversal.

Steps:

1. If the number is negative, make it positive and set a flag.
2. Extract digits and build the reversed number.
3. If the number was negative, return the reversed number as negative.

Example:

For num = -1234:

- Reverse 1234 to get 4321, then make it negative: -4321.

3-Palindrome:

Objective :

The function determines whether an integer displays reverse readability.

Steps:

1. Check for negative numbers: The function should return false for all negative numbers because they are unable to form palindromes.
2. Reverse the number: ReverseNumber serves to change numeric values into their reverse order.
3. Compare original and reversed numbers: The function should return true if the reversed number matches the original number but should return false for any other case.

Example:

For num = 121:

- Reverse 121 to get 121, which matches the original

4-Prime:

Objective :

A function exists to test integer primality when it surpasses 1 and remains divisible by 1 together with itself only.

Steps:

1. Handle small cases: When the value is equal or less than one, return false as the number is not prime. In cases where the numbers equal 2, the function should return true since 2 is a prime number.
2. Eliminate even numbers: An even number greater than 2 represents a false outcome in the decision flow-even values greater than 2 are not in the prime group.
3. Divisibility Check: In this, the code uses a loop starting from 3 to the square root of the number for checking its divisibility. In this return statement, it will detect a returned value as false whenever any divisor satisfies the condition.
4. Return True: If no divisors appear, return true to indicate the number is in fact prime.

Example:

For num = 11:

- 11 is not divisible by 2 or any odd numbers up to the square root of 11.

5-Greatest Common Divisor:

Objective :

To find the GCD between two integers you should use Euclid's algorithm.

Steps:

1. Use Euclid's Algorithm: Set b to b, but set temp to b first then divide a by b and set a to temp.
2. Repeat: Repetition of this methodology takes place until b reaches zero value.
3. Return the GCD: A will represent the GCD after the last check where b reaches zero.

Example:

For a = 56 and b = 98:

Step	Action	a	b
1	temp = b, b = a % b, a = temp	a = 98	b = 56
2	temp = b, b = a % b, a = temp	a = 56	b = 42
3	temp = b, b = a % b, a = temp	a = 42	b = 14
4	temp = b, b = a % b, a = temp	a = 14	b = 0
5	Return the GCD (a = 14)	GCD = 14	

6-Least Common Multiple:

Objective :

Identify the smallest common multiple that exists between two numbers.

Steps:

1. Find the GCD: Use the gcd function to find the greatest common divisor (GCD) of a and b.
2. Compute the LCM: Use the formula: LCM(a,b)=a×b/GCD(a,b).
3. Return the LCM: Return the calculated LCM.

Example:

For a = 12 and b = 15:

- Compute the GCD of 12 and 15: gcd(12, 15) = 3.
- Apply the LCM formula: LCM(12,15)=12×153=60\text{LCM}(12, 15) = \frac{12 \times 15}{3} = 60LCM(12,15)=312×15=60

7-Factorial:

Objective :

A program computes the factorial value of any given integer N.

Steps:

1. Initialize the result: Set fac to 1, which will hold the result.
2. Use a loop: Loop through numbers from 1 to N and multiply fac by each number.
3. Return the result: Once the loop is complete, return the value of fac.

8-Even/Odd:

Objective :

Check if a given integer is even.

Steps:

1. Check divisibility by 2: The modulus operator (%) helps determine if a number N can be divided by 2.
2. Return result: The algorithm determines that N is even if N divided by 2 produces an outcome of 0. Otherwise it classifies N as an odd number.

Example:

For N = 4:

- 4 % 2 == 0, so the number is even.

Intermediate operations on numbers

1-Prime Factorization:

Objective :

Output all prime factors found within a given integer value N.

Steps:

1. Handle even numbers: As long as N remains even we can continuously divide it by two followed by each print displaying 2 as a prime factor.
2. Handle odd factors: When working with odd numbers from 3 to the current value check if N can be divided without remainder. You can divide N by that number then display it as a prime factor when divisibility occurs.
3. Check if any remaining factor is prime: The program should display N as a prime factor only after loop completion when N exceeds 2 because it holds prime characteristics.

Example:

For N = 56:

1. Divide by 2 until Nbecomes odd:
   - 56 / 2 = 28
   - 28 / 2 = 14
   - 14 / 2 = 7 (stop, as 7 is odd)
   - Prime factor 2 appears 3 times.
2. Now, check for odd factors:
   - 7 is a prime number, so it's printed.

2-Armstrong Number:

Objective :

It checks whether the given integer N is an Armstrong number, also called Narcissistic numbers. An Armstrong number depicts any number that is equal to the total power of each digit value with a power determined by the digit count.

Steps:

1. Count the digits: Determine the number of digits in N (using NumDigits(N)).
2. Calculate the Armstrong sum: For each digit of N, raise it to the power of the number of digits and sum the results.
3. Check the Armstrong condition: If the sum equals N, return true (it's an Armstrong number), otherwise return false.

Example:

For N = 153:

1. Count the digits: 153 has 3 digits.
2. Calculate the Armstrong sum:
   - 13=11^3 = 113=1
   - 53=1255^3 = 12553=125
   - 33=273^3 = 2733=27
   - Sum = 1+125+27=1531 + 125 + 27 = 1531+125+27=153
3. Check if the sum equals N: Since the sum is equal to 153, return true.

3-Fibonacci Sequence:

Objective :

An algorithm generates and prints the first n numbers which appear in the Fibonacci sequence.

Steps:

1. Initialize variables: The starting values take s = 0 to represent the first Fibonacci number and z = 1 to represent the second Fibonacci number.
2. Use a loop to generate the sequence: Show Fibonacci numbers from 1 to n through printing each number. Next step is to print out numbers before moving onto the generation of upcoming Fibonacci numbers.
3. Print formatting: The numbers in output sequences require commas for separation and the terminal display will terminate with a period (.).

Example:

For n = 5:

1. Start with s = 0 and z = 1.
2. Print Fibonacci numbers:
   - First number: 1
   - Second number: 1
   - Third number: 2
   - Fourth number: 3
   - Fifth number: 5

4-Sum of Divisors:

Objective :

Determine how all divisors of a specific integer N (excluing N itself) total up.

Steps:

1. Initialize the sum: The variable sum becomes 0 for storing dividend sum values.
2. Loop through possible divisors: Count from 1 to N/2 because numbers beyond N/2 can only divide N by itself or N.
3. Check for divisibility: Add i to the sum when N divides with i which means N % i == 0.
4. Return the sum: Return the computed sum after executing the loop.

Example:

For N = 12:

- The divisors of 12 (excluding 12 itself) are 1, 2, 3, 4, 6.
- Sum = 1 + 2 + 3 + 4 + 6 = 16.

5-Perfect Number:

Objective :

User input must provide one integer value N to confirm its perfection as a number. Under mathematical definitions perfect numbers are numbers whose proper divisors - divisors below the number itself - sum up to create the complete number value.

Steps:

1. Find the sum of divisors: Use the sumDivisors function to calculate the sum of divisors of N (excluding N itself).
2. Check if the sum equals N: If the sum of divisors equals N, return true (it is a perfect number), otherwise return false.

Example:

For N = 6:

- The divisors of 6 (excluding 6) are 1, 2, 3.
- Sum = 1 + 2 + 3 = 6.
- Since the sum equals 6, it is a perfect number.

6-Magic Number:

Objective :

Determine whether a provided integer N holds magic number status. A magic number stands as a number which through continual summation of its digits reaches the value of 1.

Steps:

1. Initialize the sum: The variable sum receives assignment N to function as a digit sum accumulator throughout each iteration.
2. Repeat until the sum is a single digit:
3. Build the total of the digits from the sum value.
4. A check should be performed for a single-digit value if sum reaches this condition.
5. Return result: A value of true indicates a "magic number" if the final single-digit sum reaches 1 yet a value of false indicates otherwise.

Example:

For N = 19:

1. Start with sum = 19.
2. First iteration:
   a. Sum of digits of 19: 1+9=101 + 9 = 101+9=10
   b. sum = 10
3. Second iteration:
   a. Sum of digits of 10: 1+0=11 + 0 = 11+0=1
   b. sum = 1
4. Since sum is now 1, return true.

7-Automorphic Number:

Objective :

This function analyzes if a given integer value called num functions as an automorphic number. Shift to explain what makes numbers automorphic and how to detect them.

Steps:

1. Calculate the square of the number: squ = num * num.
2. Determine the number of digits in num:
3. To determine the number of digits in num begin counting from 1 while repeating divisions by 10.
4. Based on the number of digits present in num the digit variable will ultimately achieve a value composed of 10 raised to the power k.
5. Check if the last digits of the square match num:
6. An automorphic number is present when digit = num is met.

Example:

For num = 25:

1. The square of 25 is 625.
2. The number of digits in 25 is 2 (since 25 has 2 digits).
3. Check if the last 2 digits of 625 are 25:
   - 625 % 100 = 25.
4. Since the result is 25, it is an automorphic number.

Advanced operations on numbers

1-Binary Conversion:

Objective :

A function transforms integer value num into binary form while displaying the result.

Steps:

1. Handle negative numbers: During processing print the negative sign first and convert numbers to positive form.
2. Loop through the number: The division-by-2 method enables conversion of the number into binary form.
3. To get the binary digits use the identity (num % 2) which returns the division remainder.
4. The binary result needs multiplication by 10 followed by binary digit addition to the result.
5. Repetition of number division by 2 must continue until num reaches zero value.
6. Print the result: When the loop finishes execute the print statement to show the resulting binary number.

Example:

For num = 5:

1. 5 in binary is 101:
   - 5 % 2 = 1, so the last binary digit is 1.
   - 5 / 2 = 2.
   - 2 % 2 = 0, so the next binary digit is 0.
   - 2 / 2 = 1.
   - 1 % 2 = 1, so the next binary digit is 1.
   - The binary number is 101.

2-Narcissistic Number:

Objective :

A program checks whether a provided integer value of num represents a narcissistic number. A narcissistic number (sometime referred to as an Armstrong number) consists of digits which when raised to the power of the amount of digits match the original number.

Steps:

1. Find the number of digits: Check how many digits exist in num by applying NumDigits function.
2. Calculate the narcissistic sum:
3. Each digit in num gets raised at its power level equivalent to the number of digits before we add up the results.
4. The modulo operator (% 10) extracts the trailing number while temp /= 10 removes that trailing digit from the number.
5. Compare the sum with the original number: The algorithm confirms that num operates as a narcissistic number when the digit powers ordered by number distribution produce an outcome matching num.

Example:

For num = 153:

1. The number of digits in 153 is 3.
2. The sum of digits raised to the power of 3:
   - 13+53+33=1+125+27=1531^3 + 5^3 + 3^3 = 1 + 125 + 27 = 15313+53+33=1+125+27=153.
3. Since the sum equals the original number (153), it is a narcissistic number.

3-Square Root Calculation:

Objective :

This function utilizes Newton's method to estimate the square root of given number num.

Steps:

1. Start with an initial guess x=num/2.0.
2. Use Newton's method to update x=x+numx/x/2
3. Return x as the approximate square root.

Example:

Number	**Iterations**	**Output**
16	8 → 5 → 4.1 → 4.0	4.0
20	10 → 6 → 4.67 → 4.472	4.472

4-Exponentiation:

Objective :

In cases with number bases raised to integer exponents we can figure out their respective power values

Steps:

1. If exp=0, return 1.0 (any number raised to 0 is 1).
2. If exp>0, multiply the base by itself times.
3. If exp<0, multiply the base by itself times, and return its reciprocal.

Example:

Base	Exponent	Execution	Result
222	333	2×2×22 \times 2 \times 22×2×2	8.08.08.0
222	−2-2−2	1/(2×2)1 / (2 \times 2)1/(2×2)	0.250.250.25

5-Happy Number:

Objective :

To determine if a number **n** is a "happy number." A happy number repeatedly sums the squares of its digits until the result is 1 (happy) or enters a cycle ending in 4 (unhappy).

Steps:

1. Initialize nnn.
2. While n≠1n \neq 1n=1 (happy) and n≠4n \neq 4n=4(unhappy):Replace nnn with the sum of the squares of its digits.
3. Return true if n=1n = 1n=1; otherwise, return false.

Example:

Number	Execution	Result
19	12+92=821^2 + 9^2 = 8212+92=82, 82+22=688^2 + 2^2 = 6882+22=68, 62+82=1006^2 + 8^2 = 10062+82=100, 12+02+02=11^2 + 0^2 + 0^2 = 112+02+02=1	**Happy**
4	42=164^2 = 1642=16, 12+62=371^2 + 6^2 = 3712+62=37, 32+72=583^2 + 7^2 = 5832+72=58, 52+82=895^2 + 8^2 = 8952+82=89 ... (cycle ends at 4)	**Not Happy**

6-Abundant Number:

Objective :

The number category includes Abundant math types where total divisor numbers surpass the original value.

Numbers classified as Deficient show an even total of their divisors that remains lower than their original value.

Steps:

1. Calculate the sum of the proper divisors of the number.
2. Compare the sum to the number:

- If the sum is greater, the number is **Abundant**.
- If the sum is smaller, the number is **Deficient**.

Example:

Number	Sum of Divisors	Classification
121212	1+2+3+4+6=161 + 2 + 3 + 4 + 6 = 161+2+3+4+6=16	**Abundant**
151515	1+3+5=91 + 3 + 5 = 91+3+5=9	**Deficient**

7-Deficient Number:

Objective :

calculate the sum of even Fibonacci numbers up to a given limit n.

Steps:

1. Initialize the first two Fibonacci numbers (s=0,z=1).
2. Generate the Fibonacci sequence until z exceeds n.
3. Add z to the sum if it is even.
4. Return the total sum after the loop completes.

Example:

1. **Step**	1. **Fibonacci Sequence**	1. **Even Number**	1. **Sum**
1. 1	1. 0, 1	1. —	1. 0
1. 2	1. 0, 1, 1	1. —	1. 0
1. 3	1. 0, 1, 1, 2	1. 2	1. 2
1. 4	1. 0, 1, 1, 2, 3	1. —	1. 2
1. 5	1. 0, 1, 1, 2, 3, 5	1. —	1. 2
1. 6	1. 0, 1, 1, 2, 3, 5, 8	1. 8	1. 10

8-Harshad Number:

Objective :

A algorithm determines whether any number belongs to the Niven/Harshad number category or not. A Harshad number is a number which its integer value divided by the total sum of its digits yields another integer.

Steps:

1. Calculate the sum of the digits of the given number.
2. Check if the number is divisible by the sum of its digits:
- If divisible, it is a Harshad number.
- Otherwise, it is not.

3.Return the result.

9-Catalan Number:

Objective :

To calculate the nnn-th Catalan number using its closed-form formula:

$$C_n=(2n)!/(n+1)!\cdot n!$$

The algorithm uses a simplified iterative formula to avoid factorial computation.

Steps:

1. Initialize C=1.
2. Use a loop from i=1 to n: $C=C*2(2i-1)/i+1$
3. Return C, the n-th Catalan number.

Example:

n	**Catalan Number (C_n)**
0	1
1	1
2	2
3	5

10-Pascal Triangle:

Objective :

To generate and display the first n rows of Pascal's Triangle, where each entry is calculated using the binomial coefficient formula.

Steps:

For i from 0 to n:

- Print spaces for alignment.
- For j from 0 to i:
    - Calculate the binomial coefficient using: **C(i,j)=j! * i! / j!*(i−j)!**
    - Print the coefficient.

Move to a new line after each row.

Example:

Row	**Output**
Row 0	1
Row 1	1 1
Row 2	1 2 1
Row 3	1 3 3 1
Row 4	1 4 6 4 1

11-Bell Number:

Objective :

To compute the **Bell number** B_n, which represents the total number of ways to partition a set of nnn elements.

Steps:

1. Initialize bell=0 to accumulate the result.
2. For i=0 to n, add the Stirling number S(n,i) to bell.
3. Finally, return the accumulated Bell number.

Example:

For n=3, the Bell number is calculated as:

Step	**Stirling Number** S(3,i)	**Sum**(Bell number)
i=0	S(3,0)=0	0
i=1	S(3,1)=1	1
i=2	S(3,2)=3	4
i=3	S(3,3)=1	5

12-Kaprekar Number:

Objective :

The algorithm determines if a specified number qualifies as a Kaprekar number or not. Every Kaprekar number produces its original value by summing the left and right parts of its squared result.

Steps:

1. Square the number: $x=num^2$.
2. Split the square: Divide x into two parts (left and right).
3. Check if the sum of left and right equals the number.

Example:

For num=45:

- $x=45^2=2025$
- Left = 20, Right = 25
- 20+25=40 so 45 is a **Kaprekar number**.

13-Smith Number:

Objective :

The purpose of this function lies in verifying if the input number qualifies as a Smith number or not. A Smith number is a composite number (not prime) that satisfies a specific property: A number qualifies as a Smith number when the total digit value of the original number matches the aggregate digit value of its unique prime factors except the number itself.

Steps:

1. Prime check: If the number is prime, it's not a Smith number.
2. Sum of digits of the number: Calculate the sum of digits of the original number.
3. Prime factorization: Factorize the number into primes and calculate the sum of digits of each prime factor.
4. Comparison: If the sum of digits of the original number equals the sum of digits of the prime factors, it's a Smith number.

Example:

For num=666:

- 666=2×3×3×37
- Sum of digits of 666 = 18
- Sum of digits of prime factors: 2+3+3+(3+7)=18
- Since the sums match, **666 is a Smith number**.

14-Sum of Prime Numbers:

Objective :

SumOfPrimes functions with the purpose of obtaining the total value of prime numbers between 2 and the value n. The function examines each integer from 2 through n as part of its process to determine prime number status. The sum function receives prime numbers.

Steps:

1. Initialize sum: Start with a sum of 0.
2. Iterate through numbers from 2 to n: Loop through all integers starting from 2 up to n (since 1 is not a prime number).
3. Check primality: For each number, check if it is prime using the isPrime function. If the number is prime, add it to the sum.
4. Return the result: Once the loop is complete, return the total sum of prime numbers.

Example:

For n = 10, the prime numbers up to 10 are 2, 3, 5, and 7. The sum of these primes is: 2+3+5+7=172 + 3 + 5 + 7 = 172+3+5+7=17

15-Sum of Prime Factors:

Objective :

The function sumPrimeFactors calculates the sum of the digits of all prime factors of a given number num. It identifies all the prime factors of num, adds up their digits, and returns the total sum.

Steps:

1. Initialize sum: Set sum = 0.
2. Find prime factors: Loop through numbers from 2 to num to find and divide out prime factors.
3. Sum digits of factors: Add the sum of digits of each prime factor to sum.
4. Return the total sum.

Example:

For num = 18, prime factors are 2 and 3, and their digit sums are 2 and 3. Total sum is 8.

Operations on arrays

Basic Array Functions:

1- Initialize Array

Objective :

The **initializeArray** function initializes all elements of a given array to a specified value.

Steps:

1. The function takes three parameters:
2. arr[]: The array to be initialized.
3. size: The size of the array.
4. value: Each element needs a particular value assignment within the array.
5. The for loop starts its operation from the initial index that reaches up to size - 1.
6. The function sets arr[i] to this defined value during its execution cycle.

Examples:

Input	Output	Explanation

arr = [0, 0, 0, 0]
size = 4
value = 5
	arr = [5, 5, 5, 5]	All elements of the array are set to 5.

2- Print Array

Objective :

PrintArray functions enable users to display array contents on one console line through space-separated formatting.

Steps:

1. The function takes two parameters:
2. arr[]: The array to be printed.
3. size: The number of elements in the array.
4. A for loop runs during iteration to examine the array from positions 0 to size -1.
5. Each time through the loop the program prints arr[i] to console with printf alongside a space between all its iterations.

Examples:

Input	Output	Explanation

arr = [1, 2, 3, 4]
size = 4
	1 2 3 4	The function iterates through the array and prints each element separated by spaces, then ends with a newline.

3-Find Max

Objective :

The findMax function selects the largest integer element across an input array.

Steps:

1. A variable called max must contain the starting array value (arr[0]).
2. A for loop must iterate all elements in the array starting with index i set to two but stopping just before the final array position i = size - 1.
3. For each element arr[i], check if it is greater than the current value of max:
4. If true, update max to arr[i].
5. At the completion of the loop mechanism return the max variable which becomes the array's largest element.

Examples:

Input	**Output**	**Explanation**

arr = [3, 5, 7, 2, 8]
size = 5
	8	The maximum value in the array is 8.

4- *Find Min*

Objective :

As part of its process findMin identifies the most minimal value from a provided array of integers.

Steps:

1. At the beginning set min to equal the first array element (arr[0]).
2. A for loop should traverse the array from index 1 to size - 1.
3. For each element arr[i], check if it is smaller than the current value of min:
4. If true, update min to arr[i].
5. After completing the loop the function should return min which represents the smallest array entry.

Examples:

Input	Output	Explanation

arr = [3, 5, 7, 2, 8]
size = 5
	2	The minimum value in the array is 2.

5-Sum Array

Objective :

The sumArray function takes an integer array as an input to compute its complete total element value.

Steps:

1. The creation of a collection variable named sumarry performs total value tracking.
2. The processing takes place through the array using a for loop which operates from index 0 to size - 1.
3. During each loop execution period store the current array value arr[i] into sumarry.
4. Post-loop execution the sumarry variable will report the total collection of array array elements.

Examples:

Input	Output	Explanation

arr = [1, 2, 3, 4, 5]
size = 5
	15	The sum of the elements is 1+2+3+4+5=151 + 2 + 3 + 4 + 5 = 151+2+3+4+5=15.

6- Average Array

Objective :

.Through the averageArray function users can calculate the mean of all numbers present in a provided integer array.

Steps:

1. Sum the Elements:
2. After the function call sumArray executes a full summing of all array elements.
3. Sumarry is a double variable that can process an accurate calculation result.
4. Calculate the Average:
5. To get the average divide summary by value size.
6. The function stores the calculated result in averageArray.
7. Return the Result:
8. Function output delivers the averageArray data with executed calculations.

Examples:

Input	**Output**	**Explanation**

arr = [2, 4, 6, 8, 10]
size = 5
	6.0	The sum of the elements is 2+4+6+8+10=302 + 4 + 6 + 8 + 10 = 302+4+6+8+10=30, and the average is 30/5=6.030 / 5 = 6.030/5=6.0.

7- IsSorte

Objective :

The IsSorted method accepts instructions from arrays which generate accurate results as elements build specific ascending order relationships.

Steps:

1. Define a boolean value s to be true.
2. Run a while loop to advance through all array entries. Stop condition is activated either when unsorting of the array is encountered or when the end point of the array is reached.

3. In each loop:
4. orrently compares the current list element with its successive item.
5. The loop should terminate with s being set to false, and process stops there.
6. Return the value of s.

Examples:

Input	**Output**	**Explanation**

arr = [1, 2, 3, 4, 5]
size = 5
	true	All elements are in ascending order, so the function returns true.

Intermediate Array Functions

1-Reverse Array

Objective :

Function reverseArray creates the reversed copy of input array and then prints its elements.

Steps:

1. Define a new array, arr2 with the same size as that of an input array.
2. For every index i of input array, For loop is used.
3. Copy element from index size - i - 1 in input array to index i in arr2.
4. Print the values of arr2 as they get filled.

Examples:

Input	**Output**	**Explanation**

arr = [1, 2, 3, 4, 5]
size = 5
	5 4 3 2 1	The function prints the elements of the array in reverse order.

2-Count Even and Odd :

Objective :

The countEvenOdd function processes array data to determine total even and odd count occurrences which it stores through references to evenCount and oddCount.

Steps:

1. The two pointers named evenCount and oddCount serve to store the quantities of both even and odd numbers.
2. Use a for loop to iterate through each element of the array:
3. The function will increase the value stored in *evenCount if arr[i] contains 2 as a factor which can be verified when arr[i] % 2 == 0.
4. Otherwise, increment the value at *oddCount.
5. Function parameters consisting of pointers grant the capability to modify count values.

Examples:

Input	evenCount	oddCount	Explanation

arr = [1, 2, 3, 4, 5]
size = 5
	2	3	There are two even numbers (2, 4) and three odd numbers (1, 3, 5).

3-Second Largest :

Objective :

The secondLargest function detects the second largest integer from an integer array selection.

Steps:

1. findMax functions as an array search for the array's maximum element value.
2. The secondlargest variable should start with an initial small value set to -1.
3. Iterate through the array:
4. The secondlargest value gets updated when the element surpasses the current secondlargest value yet differs from max.

5. Return secondlargest.

Examples:

1. **Input**	1. **Output**	1. **Explanation**
1. arr = [1, 2, 3, 4, 5] size = 5	1. 4	1. The largest element is 5. The second largest is 4.

4-Element Frequency :

Objective :

This function measures the frequency distribution of distinct elements across an array then it prints each result.

Steps:

1. The findMin and findMax functions should identify both minimum (min) and maximum (max) values present in the array.
2. Iterate from min to max.
3. A nested loop counts the number of occurances for each value found in the array.
4. Print numbers with greater than zero count occurrences.

Examples:

Input	**Output**	**Explanation**

arr = [1, 2, 2, 3, 3, 3]
size = 6
	The frequency of element 1 is: 1
	The frequency of element 2 is: 2
	The frequency of element 3 is: 3
		Counts and prints the occurrences of each element.

5-Remove Duplicates :

Objective :

The removeDuplicates function removes duplicated elements from arrays while returning the current array size

Steps:

1. The array processing operation starts with element i in the loop.
2. For each element, compare it with subsequent elements (k):
3. The program checks for duplicate elements. When a match occurs it moves succeeding elements leftwards.
4. Second check the element with readjusted index by decreasing array size and adjusting newsize-.
5. Return the updated size (newsize).

Examples:

Input	**Output**	**New Size**	**Explanation**

arr = [1, 2, 2, 3, 3, 3]
size = 6
	arr = [1, 2, 3]	3	Duplicates are removed, leaving only unique elements.

6-binarySearch:

Objective :

A sorted array needs this method to locate the index position of specified target values efficiently.

Steps:

1. The algorithm sets low to 0 and high to size - 1.
2. While low <= high:
3. Godocument mid = (high + low) / 2.
4. The function should return mid when arr[mid] matches the target value.
5. Set low to mid + 1 when arr[mid] shows value below target.
6. Arr[mid] larger than target value sets high to mid - 1.
7. The function will return -1 anytime the target element fails to appear in the insertrange.

Example :

Step	low	high	mid	arr[mid]	Comparison	Action
1	0	4	2	6	arr[mid] == target	Return 2

7-linear Search :

Objective :

The process uses sequential element checking to determine the index position of a target value in an array.

Steps:

1. Work through the array beginning from index 0 into index size - 1.
2. For each element arr[i]:
3. A return of i (target index) will occur when arr[i] matches target.
4. The function returns -1 when the loop finishes execution without detectiung

Example for linearSearch:

Input:

- CopierModifier
- arr = [4, 7, 2, 9];
- size = 4;
- target = 7;

Execution Steps :

Index (i)	arr[i]	Comparison	Action
0	**4**	**arr[i] != target**	**Continue**
1	**7**	**arr[i] == target**	**Return 1**

Output:

- Index: 1 (Target 7 is found at index 1).

8-Left Shift :

Objective :

The array elements move left by user-defined rotations while their final positions wrap around to the beginning of the array.

Steps:

1. Calculate effective rotations: rotations = rotations % size.
2. The first rotations elements from the array get moved to a temporary storage block.
3. Shift the remaining elements left.
4. Store temporary array elements at the list's conclusion.

Example:

Input:
arr = [1, 2, 3, 4, 5], size = 5, rotations = 2.

Execution:

1. Copy: arr2 = [1, 2].
2. Shift Left: arr = [3, 4, 5, 4, 5].
3. Append: arr = [3, 4, 5, 1, 2].

Output:
arr = [3, 4, 5, 1, 2].

9-Right Shift :

Objective :

A sequence of array element shifting occurs rightward by defined rotation count with the shifted elements ending at the initial array position.

Steps:

1. Calculate effective rotations: rotations = rotations % size.
2. The final elements of the rotation will be transferred to a storage array.
3. Shift the remaining elements right.
4. Put stored temporary array elements into the beginning position.

Example:

Input:
arr = [1, 2, 3, 4, 5], size = 5, rotations = 2.

Execution:

1. Copy: arr2 = [4, 5].
2. Shift Right: arr = [1, 2, 3, 3, 4].
3. Place: arr = [4, 5, 1, 2, 3].

Output:

arr = [4, 5, 1, 2, 3].

10-Bubble Sort :

Objective :

An array sort technique uses adjacent element swaps in case they need reordering to achieve ascending order.

Steps:

1. The outer loop (i) counts the number of passes which spans up to size - 1.
2. The inner loop (j) performs consecutive element comparisons throughout the unsorted section of the array.
3. If arr[j] > arr[j+1], swap the elements.
4. The sorting process ends when the whole array becomes organized.

Example:

Input:
arr = [5, 3, 2, 4, 1], size = 5.

Execution:

Pass	**Comparison**	**Array State**
1	(5 > 3), (5 > 2), (5 > 4), (5 > 1)	[3, 2, 4, 1, 5]
2	(3 > 2), (3 > 4), (4 > 1)	[2, 3, 1, 4, 5]
3	(2 > 3), (3 > 1)	[2, 1, 3, 4, 5]
4	(2 > 1)	[1, 2, 3, 4, 5]

Output:
arr = [1, 2, 3, 4, 5].

12-selectionSort:

Objective :

An array sorting operation for ascending order selection requires finding the smallest unsorted element for placement at its array start.

Steps:

1. Loop through the array (i for passes).
2. During every pass locate the minimum index (min) within the unsorted range.
3. Switch the first unsorted element with the smallest element whenever necessary.
4. Continue all operations until the array reaches complete order.

Example:

Input:
arr = [5, 3, 2, 4, 1], size = 5.

Execution:

Pass	Action	Array State
1	Swap 5 and 1	[1, 3, 2, 4, 5]
2	Swap 3 and 2	[1, 2, 3, 4, 5]
3	No swap (already sorted)	[1, 2, 3, 4, 5]
4	No swap (already sorted)	[1, 2, 3, 4, 5]

Output:
arr = [1, 2, 3, 4, 5].

13-insertionSort:

Objective :

The process of array sorting in ascending order functions through continuous element placement according to correct position relations in relation to previously sorted array segments.

Steps:

1. Start from the second element (i = 1) since the first element is already sorted.
2. Compare the current element (arr[i]) with elements in the sorted portion.
3. Use a helper function swap to swap elements if they are in the wrong order.
4. Repeat until all elements are sorted.

Example:

Input:
arr = [5, 3, 2, 4, 1], size = 5.

Execution:

Pass	Array State	Comparison/Action
1	[3, 5, 2, 4, 1]	Compare 5 and 3, swap
2	[2, 3, 5, 4, 1]	Compare 5, 3 with 2, swap repeatedly
3	[2, 3, 4, 5, 1]	Compare 5 and 4, swap
4	[1, 2, 3, 4, 5]	Compare 5, 4, 3, 2 with 1, swap repeatedly

Output:
arr = [1, 2, 3, 4, 5].

Advanced Array Functions :

1-Find Missing Number :

Objective :

A primary purpose of this algorithm exists to locate the single uninitialized value within an ordered numerical array that extends between values 1 through n.

Steps:

1. Calculate the sum of numbers from 1 to n using the formula sum=n×(n+1)/2
2. Calculate the sum of elements in the array.
3. Find the missing number by subtracting the sum of the array from the total sum: missingnumber=sum−sumarry

Example:

For arr = [1, 2, 4, 5, 6] (size = 5):

Step	Action	Result
1	Calculate the total sum 212121 for numbers 1 to 6.	Sum = 21
2	Sum of array: 1+2+4+5+6=181 + 2 + 4 + 5 + 6 = 181+2+4+5+6=18.	Sum of array = 18
3	Missing number = 21−18=321 - 18 = 321−18=3.	Missing number = 3

2-find Pairs With Sum:

Objective :

The main purpose of find Pairs With Sum functions to discover and display one-by-one all unique arrays combinations which amount to a designated target sum.

Steps:

1. Iterate through the array with two loops: A first loop chooses an element for examination which is followed by a second loop that compares that element to all subsequent elements.
2. Check if the sum of the pair equals the target sum.
3. Display the pair when the sum of selected values reaches the target value.

Example:

For arr = [1, 3, 2, 4, 6] and sum = 6:

Pair	Sum Check	Result
(1, 3)	1 + 3 = 4	No
(1, 2)	1 + 2 = 3	No
(1, 4)	1 + 4 = 5	No
(1, 6)	1 + 6 = 7	No
(2, 4)	2 + 4 = 6	Yes

Output:

(2, 4)

3-find SubArray With Sum:

Objective :

The Find SubArray With Sum algorithm locates and displays subarray sequences which match the target sum value in the array. FindSubArrayWithSum generates a message output for failing to find a sum target value in any subarray.

Steps:

1. Iterate through the array: The outer loop initializes at each element position inside the array.
2. Sum the elements: Put an inner loop in place to combine the next elements to the sum.
3. Check if the sum matches the target: Once the algorithm finds a matching subarray print it out.
4. Handle no match: The program prints a specific message when the array contains no subarray that reaches the target sum.

Example:

For arr = [1, 2, 3, 7, 5] and sum = 12:

Step	Action	Result
1	Start from index 1, sum = 12	Subarray [2, 3, 7] found
2	Print the subarray	Subarray with the given sum found between indexes 1 and 3: 2 3 7

4-Rearrange Alternate Positive Negative:

Objective :

The array should receive reorganizing treatment to create an alternation of positive and negative elements with the first position holding a positive value. A rearrangement will be performed if possible but no modifications will be made when array reordering is not feasible.

Steps:

1. Iterate through the array: Check the element located at position i.
2. Check for misplaced positive numbers: Changed Title: Swapping Positive Numbers for Better Alternation
3. Check for misplaced negative numbers: The algorithm swaps current negative index pairs if the absolute value indicates placement within even positions.
4. The procedure continues until complete alternate positioning of elements exists.

Example:

For arr = [1, -2, 3, -4, 5, -6]:

Step	Action	Array State
1	Start with index 0 (positive)	[1, -2, 3, -4, 5, -6]
2	Check index 1 (negative)	[1, -2, 3, -4, 5, -6]
3	Check index 2 (positive)	[1, -2, 3, -4, 5, -6]
4	Check index 3 (negative)	[1, -2, 3, -4, 5, -6]
5	Swap at index 2 and index 4	[1, -2, 5, -4, 3, -6]
6	Final rearranged array	[1, -2, 5, -4, 3, -6]

5-Find Majority Element :

Objective :

The find Majority Element function aims to detect the majority element in an array which exists when it appears more than half the array's size. When a majority element does not exist the function outputs an appropriate message.

Steps:

1. Iterate through the array: Use two loops to count occurrences of each element.
2. Track the element with the highest count: Update the majority element and its count.
3. Check if the count exceeds half the array size: If true, print the majority element; otherwise, print that no majority element exists.

Example:

For arr = [3, 3, 4, 2, 4, 4, 2, 4, 4]:

Step	Action	Result
1	Start with element 3, count = 2	Count 3: [3, 3]
2	Move to next element 4, count = 4	Count 4: [4, 4, 4, 4]
3	No other element exceeds count 4	Majority element = 4

6- Longes tIncrea sing Subsequence:

Objective :

How long is the longest subsequence of increasing values in this array?

Steps:

1. Iterate through the array: Two iteration loops help you check each element pair.
2. Track the subsequence length: The length should rise up when examining a next element which exceeds the present one.
3. Update the maximum length: When the current sequence length passes beyond the maximum value it establishes a new maximum length.

Example:

For arr = [10, 22, 9, 33, 21, 50, 41, 60, 80]:

Step	Action	Current LIS Length	Max LIS Length
1	Check from 10, 22, ...	6	6

7-Find Duplicates :

Objective :

The program must locate and display duplicate values found within the array. In case no duplicates exist it should present an appropriate message. A verification message will be printed when the search identifies no duplicate elements.

Steps:

1. Iterate through the array: A double loop structure should analyze every element with its complete set of other elements.
2. Check for duplicates: Display the duplicate element when two identical elements appear yet no previous items have been flagged as duplicates (k).
3. Mark the duplicate: After finding a duplicate set k to the result and execute duplicate search operations on remainder of elements.
4. Handle no duplicates: During theuplicate check, the program prints "No duplicates found" if it finds no matches.

Example:

For arr = [1, 2, 3, 2, 4, 5, 1]:

Step	Action	Result
1	Compare 1, 2, 3, ...	2 is a Duplicate
2	Compare 1, 5, 1	1 is a Duplicate
3	No other duplicates found	"No duplicates found."

8-Find Inter section:

Objective :

The program identifies common elements between two arrays then displays these elements. If no intersections exist it prints an appropriate notice. Print a notice when arrays demonstrate no intersection points between them.

Steps:

1. Compare elements: Two loops need to operate together to determine whether elements from arr1 appear in arr2.
2. Avoid duplicates: Print elements which are common elements but also mark them to prevent duplicate prints.
3. Handle no intersection: The program will display "No intersection elements found" in case no common elements exist between arrays.

Example:

For arr1 = [1, 2, 3, 4, 5] and arr2 = [3, 4, 5, 6, 7]:

Step	Action	Intersection Element
1	Compare 1 with all elements in arr2	No match
2	Compare 2 with all elements in arr2	No match
3	Compare 3 with all elements in arr2	3
4	Compare 4 with all elements in arr2	4
5	Compare 5 with all elements in arr2	5

9-Find Union :

Objective :

The program must determine and display every unique element which exists in both arrays combined.

Steps:

1. Combine the arrays: The new array unionArr contains all elements from arr1 plus all elements from arr2.
2. Remove Duplicates: The function removeDuplicates will remove the duplicated entries from the merged array.
3. Print the union: Show the unique items from the union collection.

Example:

For arr1 = [1, 2, 3, 4] and arr2 = [3, 4, 5, 6]:

Step	Action	Result
1	Combine both arrays	[1, 2, 3, 4, 3, 4, 5, 6]
2	Remove duplicates	[1, 2, 3, 4, 5, 6]
3	Print the union	1 2 3 4 5 6

Operations on matrices

1-Initialize Matrix:

Objective :

A systematic value fills every cell within a 2D matrix.

Steps:

1. Initialize sum: Set sum = 0.
2. Find prime factors: Search from number 2 to num for prime factors using a looping mechanism to divide them out.
3. Sum digits of factors: Each prime factor should contribute its digit sum to sum.
4. Return the total sum.

Example:

For num = 18, prime factors are 2 and 3, and their digit sums are 2 and 3. Total sum is 8.

2-Print Matrix:

Objective :

Present data from a 2D matrix in a labeled grid arrangement.

Steps:

Inputs:

1. rows: Number of rows in the matrix.
2. cols: Number of columns in the matrix.
3. matrix[rows][cols]: 2D array to be printed.

Outer Loop (Rows):

1. The process loops through every row in the matrix.

Inner Loop (Columns):

1. It performs a cycle through all the columns within a given row.
2. The code displays each matrix element with a new tab character between them.

New Line:

1. Each row printing sequence terminates with a newline character (\n) to achieve suitable grid presentation.

Example:

Input Matrix:	**Printed Output**:
{1, 2, 3},	**1 2 3**
{4, 5, 6},	**4 5 6**
{7, 8, 9}}	**7 8 9**

3-Input Matrix:

Objective :

User input is populated into a 2D matrix through the inputMatrix function design. During each iteration of row and column indexes the user must input values for all elements in the matrix.

Steps:

1. The program needs users to provide information about rows (rows), columns (cols) and matrix (matrix).
2. Users need to enter matrix elements according to the displayed instruction.
3. A loop system should process all rows within the matrix.
4. The current row needs two loops working together to access its columns one by one.

5. The program demands user input for matrix elements in specific positions.
6. The program adds a new line after completing each row to make formatting easier to understand.

Example :

Input:

- Number of rows: 2
- Number of columns: 3

Execution (user input):

Prompt	Input
matrix[0][0] =	1
matrix[0][1] =	2
matrix[0][2] =	3
matrix[1][0] =	4
matrix[1][1] =	5
matrix[1][2] =	6

Output Matrix:

1	2	3
4	5	6

4-Add Matrices:

Objective :

A third matrix contains element-wise additions of two submitted matrices.

Steps:

1. Loop through each row of the matrices.
2. Loop through each column in the current row.
3. Add the corresponding elements of mat1 and mat2.
4. Store the result in the result matrix.

Example:

Let’s consider two matrices mat1 and mat2:

mat1	mat2	result (mat1 + mat2)
1 2 3	4 5 6	5 7 9
4 5 6	7 8 9	11 13 15

5-Subtract Matrices:

Objective :

The function enables element-level subtraction from two matrices by outputting results to a third output matrix.

Steps:

1. Loop through each row of the matrices.
2. Loop through each column in the current row.
3. Subtract the corresponding elements of mat1 and mat2.
4. Store the result in the result matrix.

Example:

Let’s consider two matrices mat1 and mat2:

mat1	mat2	result (mat1 - mat2)
5 7 9	4 5 6	1 2 3
10 11 12	7 8 9	3 3 3

6-Multiply Matrices:

Objective :

A function operates by multiplying two matrices while placing the outcome within another matrix. To multiply matrices successfully the number of columns in the first matrix must be equal to the number of rows in the second matrix.

Steps:

1. **Check dimensions**: Ensure the number of columns in mat1 equals the number of rows in mat2. If not, multiplication is not possible.
2. **Iterate through the result matrix**:

- For each element in the result matrix, compute the dot product of the corresponding row from mat1 and column from mat2.
- Store the result of the dot product in the result matrix.

Example:

For matrices mat1 (2x3) and mat2 (3x2):

mat1	mat2
1 2 3	4 5
4 5 6	6 7

Result matrix:

result
40 46
94 109

7-ScalarMultiply Matrix:

Objective :

Change every matrix element with a single value to change its original structure.

Steps:

1. Iterate through each element: Loop through every element of the matrix.
2. Multiply by scalar: For each element, multiply it by the given scalar value.
3. Update the matrix: Store the scaled value back into the same matrix.

8-SquareMatrix:

Objective :

To determine if a matrix is square we should compare the number of rows with the number of columns.

Steps:

1. Compare the number of rows with the number of columns.
2. Return true if they are equal (i.e., a square matrix), otherwise return false.

Example:

For a 3x3 matrix:

1	2	3
4	5	6
7	8	9

The function will return true because it is a square matrix (3 rows and 3 columns).

9-Identity Matrix:

Objective :

Verify both 1s in the principal diagonal and 0s everywhere else across the matrix to identify identity matrix status.

Steps:

1. A single processing cycle analyzes every component inside the matrix.
2. A value of 1 should exist within main diagonal elements.
3. Analyzing the element reveals what happens when its row count differs from its column positioning. Any value other than zero detects an invalid symmetric matrix.

4. Evaluation results in success only when all stated conditions match but failure if even one condition is unfulfilled.

10-Diagonal Matrix:

Objective :

Testing whether a matrix meets the requirements to be classified as diagonal forms the purpose of this function. In diagonal matrices all elements outside the primary diagonal must be zero because diagonal matrices feature zero values in their off-diagonal positions.

Steps:

1. Examine every entry in the matrix through sequential processing.
2. The algorithm confirms that each matrix element is located outside the main diagonal (i != j).
3. The function returns false when any off-diagonal element is detected.
4. A return value of true indicate a diagonal matrix when all off-diagonal elements are zero.

11-Symmetric Matrix:

Objective :

The procedure checks whether a matrix meets the requirements for being a symmetric matrix. Symmetric matrix status can be determined through a simple requirement where it equals its transpose matrix therefore (i, j) and (j, i) element positions must contain equivalent values across each position combination.

Steps:

1. The program processes every component found inside the matrix.
2. During the evaluation of element (i, j) verify if it matches the value stored at position (j, i).
3. A test returns false when any pair of elements (i, j) does not match element (j, i) thus determining matrix non-symmetry.
4. The test produces true when the entire matrix displays equal corresponding elements indicating symmetry.

12-UpperTriangular:

Objective :

This algorithm tests a matrix to check whether it is an upper triangular matrix. Upper triangular matrix classification applies to matrices where all components beneath the main diagonal hold values of zero.

Steps:

1. At row two the matrix iteration process should begin because row one does not have any elements which fall under the diagonal space.
2. For each row index that starts at 1 examine all matrix[i][j] elements where j appears before i.
3. When the function detects diagonal-crossing entries it returns false to indicate non-upper triangular matrix behavior.
4. An upper triangular matrix pattern emerges when all diagonal position elements disappear (True value is returned).

13-inverse Matrix:

Objective :

We will calculate square matrix inverse using Gaussian elimination. Initial operations on the matrix incorporate an identity matrix so that the original matrix transforms into an identity matrix during which time the identity matrix evolves into its inverse form.

Steps:

1. Attach an identity matrix to the existing matrix structure.
2. Pivoting: In each row divide by the diagonal entry to get a value of one.
3. erase all numerical values that exist above or below the main diagonal.
4. Once invert the second half of the augmented matrix form.

14-Trace Matrix:

Objective :

Determining the trace value of square order matrices.

Steps:

1. The storage of diagonal element summation requires trace as a variable.
2. The program examines all row-column relationships while i matches j during evaluation.

3. During processing the diagonals receive an added value.
4. Return the calculated trace.

15-Rotate Matrix90:

Objective :

The algorithm enables square matrix rotation by 90 degrees clockwise.

Steps:

1. Transpose the Matrix:For all i < j values switch the elements stored in matrix[i][j] with elements in matrix[j][i].
2. Reverse Each Row:For a 90-degree clockwise rotation complete the sequence by reversing the pattern of each row.

Operations on strings

1-stringLength:

Objective :

A algorithm determines string length through complete character counting which stops at the null terminator (\0).

Steps:

1. Initialize Counter: Set size = 0.
2. Loop Through String: Increment size for each character until the null terminator (\0) is reached.
3. Return Length: Output the value of size.

Example:

Input String	Length
"Hello"	5
"World!"	6
""	0

2-StringConcat:

Objective :

The objective is to add one string (src) onto the end of another string (dest) but disallowing native string functions.

Steps:

1. Move to the end of dest (find the null terminator).
2. Copy each character from src to dest, advancing both pointers.
3. Add a null terminator (\0) at the end of the concatenated string.

3-Empty:

Objective :

Program use this function to check if a string field contains value.

Steps:

1. Check if the pointer str is NULL (string is not initialized).
2. If not NULL, check if the first character is the null terminator '\0'.
3. Return true if either condition is true; otherwise, return false.

Example:

Input	Output
str = ""	true
str = "Hello"	false
str = NULL	true

4-ReverseArray:

Objective :

The algorithm enables in-place reversal of array elements.

Steps:

1. Check if the array is NULL or size ≤ 0; exit if true.
2. Use two pointers (start and end) to swap elements.
3. Increment start and decrement end until they meet.

Example:

Input	Size	Output
{'a', 'b'}	2	{'b', 'a'}
{'1', '2', '3'}	3	{'3', '2', '1'}

5-UpperCase:

Objective :

The algorithm enables in-place reversal of array elements.

Steps:

1. Start at the first character of the string.
2. Check if the character is lowercase.
3. Convert it to uppercase using toupper().
4. Move to the next character until the end of the string ('\0') is reached.

Example

Input	Output
"hello"	"HELLO"
"MixedCase!"	"MIXEDCASE!"

6- Find Substring:

Objective :

Identify the initial position of a substring compared to its enclosure within a main string.

Steps:

1. Get Lengths: Measure the lengths of the main string and the substring.
2. Iterate: Loop through the main string.
3. Match: When the first character of the substring matches, check subsequent characters.
4. Return: If a match is found, return the starting index; otherwise, return -1.

Example

For "hello world" and substring "world", the output is 6.

7-Palindrome:

Objective :

The code determines whether a supplied string can read the same backward or forward.

Steps:

1. Length Calculation: Determine the length of the input string.
2. Reverse the String: Create a temporary array and reverse the input string.
3. Compare: Check if the original string matches the reversed string.
4. Return: If both are equal, return true; otherwise, return false.

Example

For input "madam", the output is true since it's a palindrome.

8-Anagram:

Objective :

A program tests whether two input strings qualify as anagrams.

Steps:

1. Length Comparison: If the strings' lengths are different, return false as they cannot be anagrams.
2. Character Matching: For each character in the first string, check if it exists in the second string.
3. Return: If all characters match in both strings, return true; otherwise, return false.

Example

For input "listen" and "silent", the output is true since they are anagrams

9-countWords:

Objective :

Read sentences in a string and use space breaks to count individual words.

Steps:

1. Initialize: Create a counter c to track the number of words and a flag inword to check if the current character is part of a word.
2. Traverse: Loop through each character in the string:
3. If it's not a space and we're not already inside a word, increment the word counter.
4. If it's a space, mark that we're outside a word.
5. Return: After traversing the string, return the word count.

Example

For the input string "Hello world! How are you?", the output is 5.

10-StringCopy:

Objective :

The utility performs a string data transfer procedure from source to destination files.

Steps:

1. Initialize an index variable.
2. Iterate through the source string until the null terminator.
3. Copy each character from src to dest.
4. Terminate the destination string with a null character.

Example

Let's consider an example where we want to copy the string "Hello" from src to dest.

11-ToLower:

Objective :

The application of toLowerCase functionAdvertiser Identity Verification (AIV) System changes all uppercase text to lowercase format without modification of other special characters throughout the input string.

Steps:

1. Loop Through the String: Iterate over each character until reaching the end ('\0').
2. Check Uppercase: If the character is uppercase, convert it to lowercase by adding 32 to its ASCII value.
3. Process Next Character: Move to the next character and repeat.
4. Complete Conversion: Stop when all characters are processed.

12-countVowelsConsonants:

Objective :

The program counts vowel and consonant occurrences within an entered string.

Steps:

1. Convert String to Lowercase: Use the toLowerCase function to handle case-insensitivity.
2. Initialize Counters: Set counters for vowels and consonants to zero.
3. Iterate Through the String: Traverse each character of the string.
4. Check for Letters: Verify if the character is alphabetic (a-z).
5. Classify Character:
6. If the character is a vowel (a, e, i, o, u, y), increment the vowels counter.

7. Otherwise, increment the consonants counter.
8. Print Results: Output the counts for vowels and consonants.

Example

Input: "Hello World"
Output:

1. CopierModifier
2. The number of vowels in your string is 3
3. The number of consonants in your string is 7

13-RemoveWhitespaces:

c

1. Traverse the string using an index i.
2. If a whitespace is found, shift all subsequent characters one position to the left.
3. Repeat until the end of the string.
4. The result is a string without whitespaces.

Example

Input: "Hello World"
Output: "HelloWorld"

14-charFrequency:

Objective :

Determine the frequency of each character in a given string, ignoring case and whitespaces.

Steps:

1. Convert the string to lowercase using toLowerCase().
2. Remove whitespaces from the string with removeWhitespaces().
3. For each letter (a to z):

- Count its occurrences in the string.
- If the frequency is greater than 0, print the result.

15-RemoveDuplicates:

Objective :

Remove all duplicate characters from a given string.

Steps:

1. Convert to lowercase: Ensure all characters are case-insensitive.
2. Iterate through the string: For each character, compare it with subsequent characters.
3. Check for duplicates: If a duplicate is found, shift all characters to the left to remove it.
4. Repeat: Continue until all duplicates are removed.

Example

Input:
"Programming"

Output:

```
programing
```

16-LongestWord:

Objective :

Find the longest word in a given string.

Steps:

Initialize variables:

- Track indices, lengths, and the maximum word length (maxcount).
- Add a trailing space to the string to ensure the last word is processed.

Iterate through the string:

- Detect spaces as word boundaries.
- Calculate the current word's length.
- Update the longest word if the current word is longer.

Store the longest word:

- Copy the longest word into a separate variable.

Print the result:

- Display the longest word.

Example

Input:
"I love programming in C language"

Output:

Longest is programming

17- CountChar:

Objective :

Count the occurrences of a specific character in a string.

Steps:

1. **Initialize a counter**:
2. Start with count = 0.
3. **Iterate through the string**:
4. Check each character of the string.
5. Increment the counter whenever the target character (ch) matches the current character.
6. **Return the result**:
7. After the loop finishes, return the value of count.

18- ReverseString:

Objective :

Reverse a given string and store the result in another string.

Steps:

1. Get the string length: Loop through the string to count characters.
2. Reverse the string: Copy characters from the end of the original string to the beginning of the reversed string.
3. Add null terminator: Place '\0' at the end of the reversed string.

19-int string Compare:

Objective :

Compare two strings and return the difference between their mismatched characters or lengths.

Steps:

1. Loop through the strings: Compare each character of both strings one by one.
2. Check for mismatch: If a mismatch occurs, return the difference between the characters.
3. Handle different lengths: If strings have different lengths, return the difference at the end of the strings.

Example

Input:

str1 = "hello"

str2 = "hella"

Output: 4 (since 'o' - 'a' = 4)

20-Substring:

Objective :

Extract a substring from a string based on the given start and end indices.

Steps:

1. Initialize index: Set an index to keep track of the position in the result array.
2. Loop through the string: Copy characters from the start index to the end index into the result array.
3. Null-terminate the result: After copying the characters, add the null terminator to mark the end of the substring.

Example:

Input:

str = "programming"

start = 3

end = 6

Output:

result = "gra"

Optional parts

1. Caesar Cipher

Objective:

Shift each letter in the text by a fixed number of positions in the alphabet.

Steps:

1. Take the input text and shift value.
2. Loop through each character:
   - If it's a letter, shift it by the given value while keeping it within the alphabet range.
   - Preserve uppercase and lowercase letters.
3. Return the encrypted or decrypted text.

Example:

```
char text[] = "Hello";

caesarCipher(text, 3);

printf("%s", text);
```

Output: "Khoor"

2. Atbash Cipher

Objective:

Replace each letter with its reverse counterpart in the alphabet (A ↔ Z, B ↔ Y, etc.).

Steps:

1. Loop through each character in the text.
2. If it's a letter:
   - Convert 'A' to 'Z', 'B' to 'Y', etc.
   - Convert 'a' to 'z', 'b' to 'y', etc.
3. Return the transformed text.

Example:

```
char text[] = "Hello";

atbashCipher(text);

printf("%s", text);
```

Output: "Svool"

3. Substitution Cipher

Objective:

Replace each letter in the text with a corresponding letter from a predefined key.

Steps:

1. Define a key mapping each letter to another.
2. Loop through the text:
   - Replace each letter with its corresponding key value.
3. Return the transformed text.

Example:

```
char text[] = "Hello";

char key[] = "QWERTYUIOPASDFGHJKLZXCVBNM";

substitutionCipher(text, key);

printf("%s", text);
```

Output: Itssg

4. XOR Cipher

Objective:

Encrypt text using bitwise XOR with a key.

Steps:

1. Loop through each character in the text.
2. Apply XOR operation with the key.
3. Return the encrypted or decrypted text.

5. Vigenère Cipher

Objective:

Encrypt text by shifting letters based on a repeating key.

Steps:

1. Loop through each character of the text.
2. Determine the shift value from the key.
3. Apply the shift while keeping within the alphabet range.
4. Return the transformed text.

Example:

```
char text[] = "Hello";

char key[] = "KEY";

vigenereCipher(text, key, 1);

printf("%s", text);
```

Output: Rijvs