SIGNALS AND SYSTEMS LAB (CSE-301L)

Spring 2024, 4th Semester

Lab Report 01

Submitted by**: Hassan Zaib Jadoon**

Registration Number**: 22PWCSE2144**

Section: **A**

"On my honor, as a student at the University of Engineering and Technology Peshawar, I have neither given nor received unauthorized assistance on this academic work."

Signature:

Date: 10/3/2024

Submitted To: Dr. Safdar Nawaz Khan Marwat

Department of Computer Systems Engineering
University of Engineering and Technology Peshawar

TASK 01

Write a function that accepts temperature in degrees F and computes to corresponding value in degrees C. The relation between the two is:

$$T = 5/9(T_f - 32)$$

Make sure that you test your function with three different input values.

Problem Analysis

The problem is to write a MATLAB function that converts temperature from Fahrenheit (°F) to Celsius (°C) using the formula:

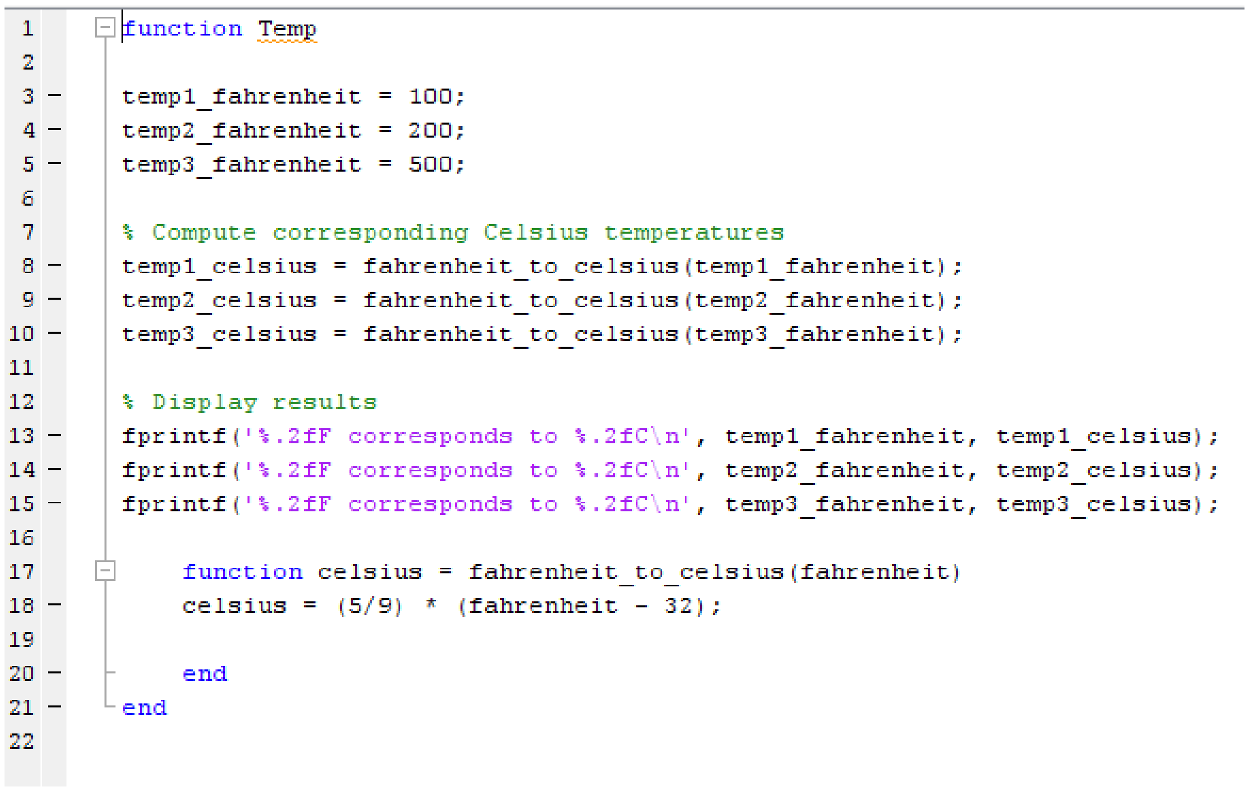
$$T_c = 5/9(T_f - 32)$$

Where T_C is the temperature in Celsius and T_f is the temperature in Fahrenheit.

Algorithm

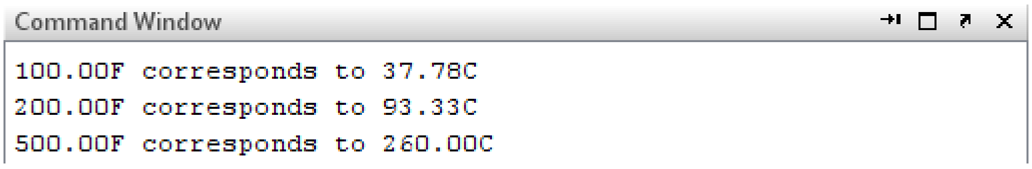
1. Start by defining a MATLAB function named Convert to that takes one input argument temp_F representing the temperature in Fahrenheit.
2. Inside the function, apply the conversion formula to calculate the temperature in Celsius.
3. Return the temperature in Celsius as the output of the function.
4. Test the function with three different input values to ensure it works correctly.

Code



```
function Temp

temp1_fahrenheit = 100;
temp2_fahrenheit = 200;
temp3_fahrenheit = 500;

% Compute corresponding Celsius temperatures
temp1_celsius = fahrenheit_to_celsius(temp1_fahrenheit);
temp2_celsius = fahrenheit_to_celsius(temp2_fahrenheit);
temp3_celsius = fahrenheit_to_celsius(temp3_fahrenheit);

% Display results
fprintf('%.2fF corresponds to %.2fC\n', temp1_fahrenheit, temp1_celsius);
fprintf('%.2fF corresponds to %.2fC\n', temp2_fahrenheit, temp2_celsius);
fprintf('%.2fF corresponds to %.2fC\n', temp3_fahrenheit, temp3_celsius);

    function celsius = fahrenheit_to_celsius(fahrenheit)
    celsius = (5/9) * (fahrenheit - 32);

    end
end

```

Output:



```
100.00F corresponds to 37.78C
200.00F corresponds to 93.33C
500.00F corresponds to 260.00C
```

Conclusion

The function successfully converts temperature from Fahrenheit to Celsius using the provided formula. Testing the function with three different input values confirms its correctness and accuracy in temperature conversion.

TASK 02

For the arrays x and y given below, write MATLAB code to find all the elements in x that are greater than the corresponding elements in y.

x = [-3, 0, 0, 2, 6, 8]

y = [-5, -2, 0, 3, 4, 10]

Problem Analysis

The problem is to write MATLAB code that finds all the elements in array x that are greater than the corresponding elements in array y. The arrays x and y are given as follows:

x = [-3, 0, 0, 2, 6, 8]

y = [-5, -2, 0, 3, 4, 10]

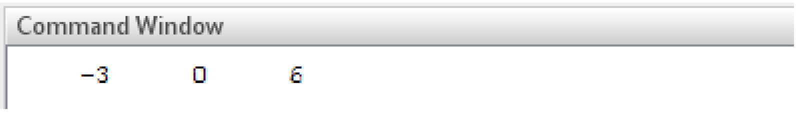
Algorithm

1. Define the arrays x and y with the given values.
2. Use array indexing and logical indexing in MATLAB to create a new array result containing elements from x that are greater than the corresponding elements in y.
3. Display the elements in result.

Code

```
x = [-3, 0, 0, 2, 6, 8];
y = [-5, -2, 0, 3, 4, 10];

% Find elements in x greater than corresponding elements in y
result = x(x > y);
disp(result);
```

Output



```
    -3     0     6
```

Conclusion

The algorithm uses logical indexing in MATLAB to efficiently find elements in array x that are greater than the corresponding elements in array y. The code provides a concise and effective solution to the problem.

TASK 03

Using the branching constructs, for $0 < a \leq 16$, find the values of C defined as follows:

$$C = 4ab \text{ for } 1 \leq a \leq 8$$
$$\mathbf{C}=ab \text{ for } 8 < a \leq 16$$

and b = 12.

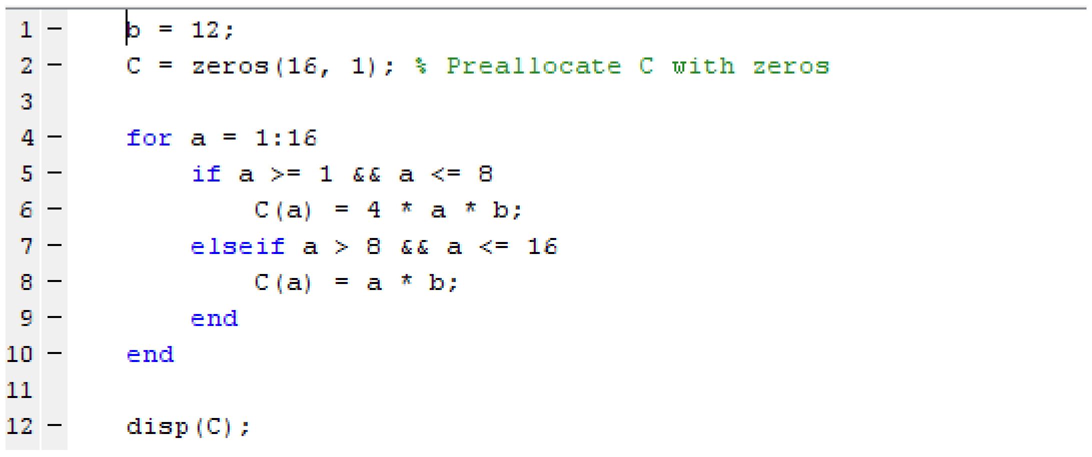
Problem Analysis

The problem involves generating a sequence of numbers that starts at -120, increments by 4 until it reaches 120, and includes both negative and positive integers. The sequence follows a pattern where each successive number differs by 4.

Algorithm

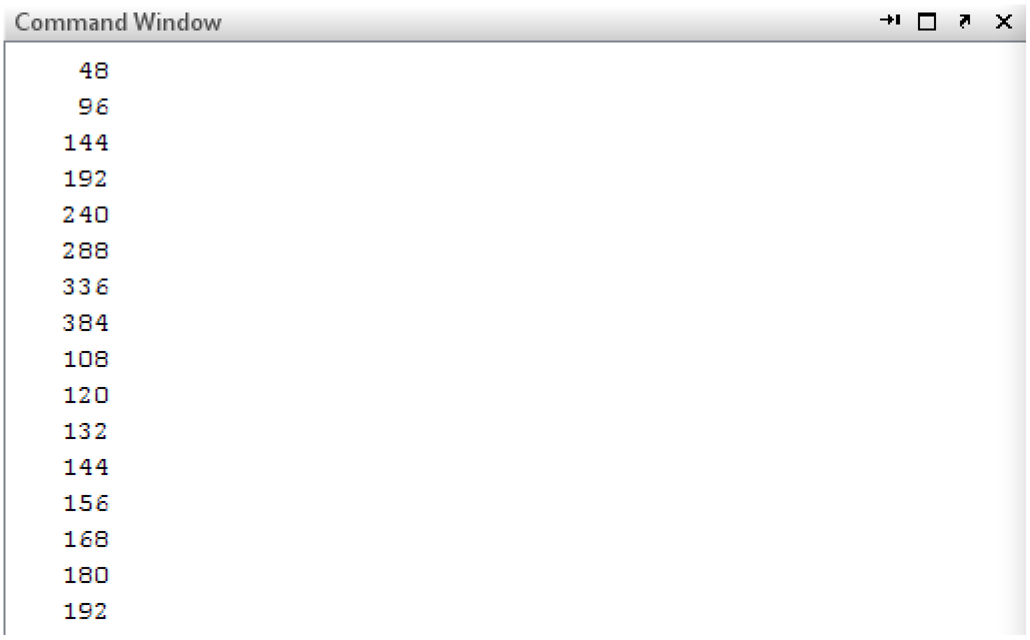
1. Initialize the value of b to 12.
2. Use branching constructs (if-else statements) to calculate the value of C based on the given conditions:
3. If a is between 1 and 8 (inclusive), calculate C as 4ab.
4. If a is between 9 and 16 (inclusive), calculate C as ab.
5. Display the calculated values of C for each value of a.

Code



```
b = 12;
C = zeros(16, 1); % Preallocate C with zeros

for a = 1:16
    if a >= 1 && a <= 8
        C(a) = 4 * a * b;
    elseif a > 8 && a <= 16
        C(a) = a * b;
    end
end

disp(C);
```

Output



```
    48
    96
   144
   192
   240
   288
   336
   384
   108
   120
   132
   144
   156
   168
   180
   192
```

Conclusion

The algorithm uses branching constructs in MATLAB (if-else statements) to calculate the value of C based on the conditions specified for different values of a.

TASK 04

For the values of integer, a going from 1 to 10, using separately the methods of branching constructs and the Boolean alternative expressions, find the values of C if:

$$C = \begin{cases} a^2 & \text{for} \quad a < 3 \\ a + 3 & \text{for} \quad 3 \le a \le 7 \\ a & \text{for} \quad a > 7 \end{cases}$$

Problem Analysis

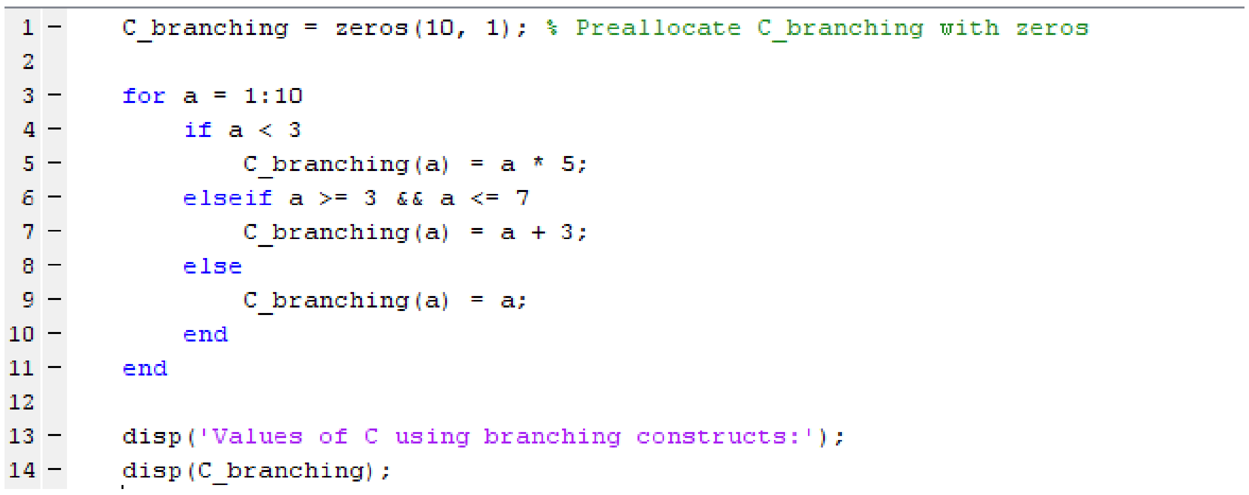
The problem is to calculate the values of C for different ranges of integer a using branching constructs in MATLAB. The conditions for calculating C are as follows:

1. If a is less than 3, C is calculated as a * 5.
2. If a is between 3 and 7 (inclusive), C is calculated as a + 3.
3. If a is greater than 7, C is equal to a.

Algorithm

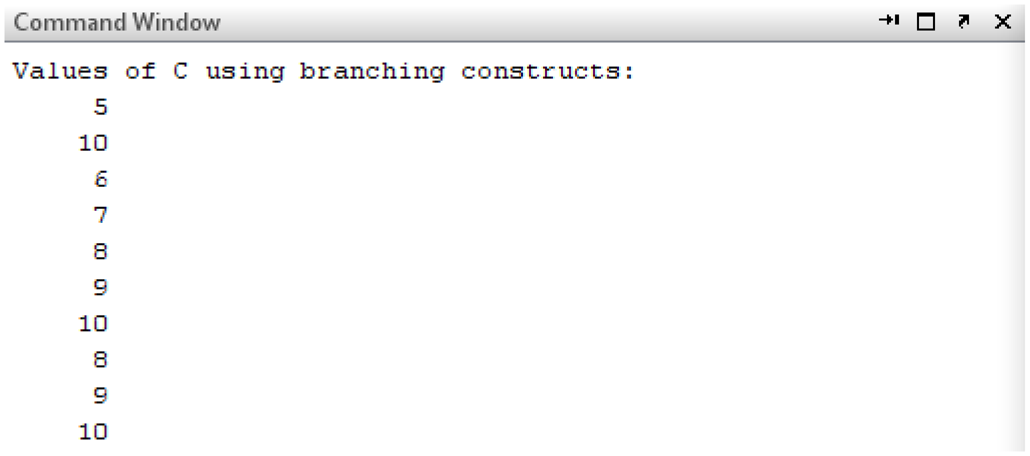
Initialize an array Branching with zeros to store the calculated values of C.
Loop through the values of a from 1 to 10.
Use branching constructs (if-else statements) to check the conditions and calculate the corresponding values of C for each value of a.
Store the calculated values of C in the Branching array.
Display the values of Branching after the loop completes.

Code



```matlab
C_branching = zeros(10, 1); % Preallocate C_branching with zeros

for a = 1:10
    if a < 3
        C_branching(a) = a * 5;
    elseif a >= 3 && a <= 7
        C_branching(a) = a + 3;
    else
        C_branching(a) = a;
    end
end

disp('Values of C using branching constructs:');
disp(C_branching);
```

Output



```
Values of C using branching constructs:
     5
    10
     6
     7
     8
     9
    10
     8
     9
    10
```

Conclusion

The MATLAB code effectively uses branching constructs to calculate the values of C based on the specified conditions for different ranges of integer a.

TASK 05

Rewrite the following statements to use only one if statement.

```
if x < y
        if z < 5
        w = x*y*z
        end
end.
```

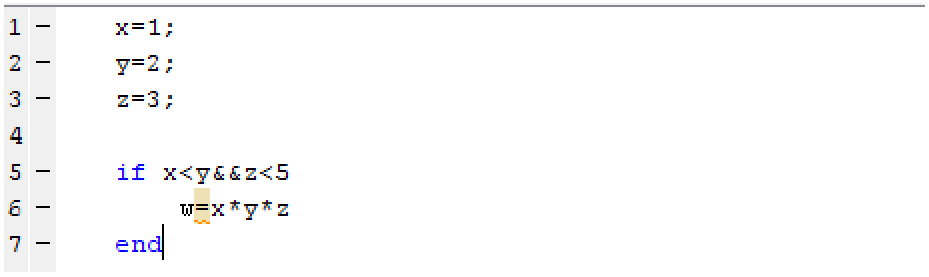
Problem Analysis

The problem is to evaluate the expression w = x * y * z if and only if both conditions x < y and z < 5 are true. The values of x, y, and z are predefined as x = 1, y = 2, and z = 3, respectively.

Algorithm

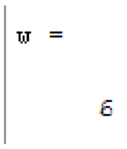
1. Assign the values x = 1, y = 2, and z = 3.
2. Check if both conditions x < y and z < 5 are true using the logical AND operator &&.
3. If both conditions are true, compute the expression w = x * y * z.
4. Display the value of w.

Code



```
x=1;
y=2;
z=3;

if x<y&&z<5
    w=x*y*z
end
```

Output



```
w =

     6
```

Conclusion

This code snippet efficiently evaluates the expression w = x * y * z only when both conditions x < y and z < 5 are satisfied, providing an optimized and concise solution to the problem.

TASK 06

Using for loop, generate the cube of the first ten positive integers.

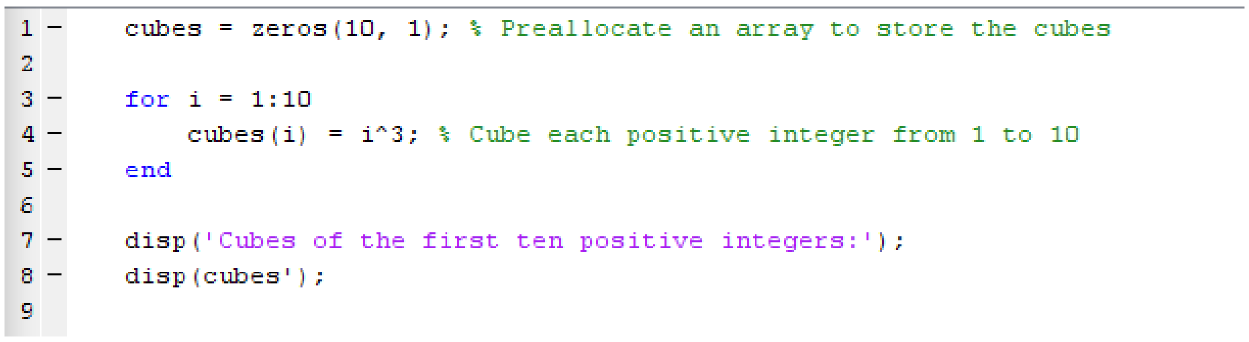
Problem Analysis

The problem is to calculate the cube of the first ten positive integers (1 to 10) using a for loop in MATLAB. The cube of a number is obtained by multiplying the number by itself twice (number * number * number).

Algorithm

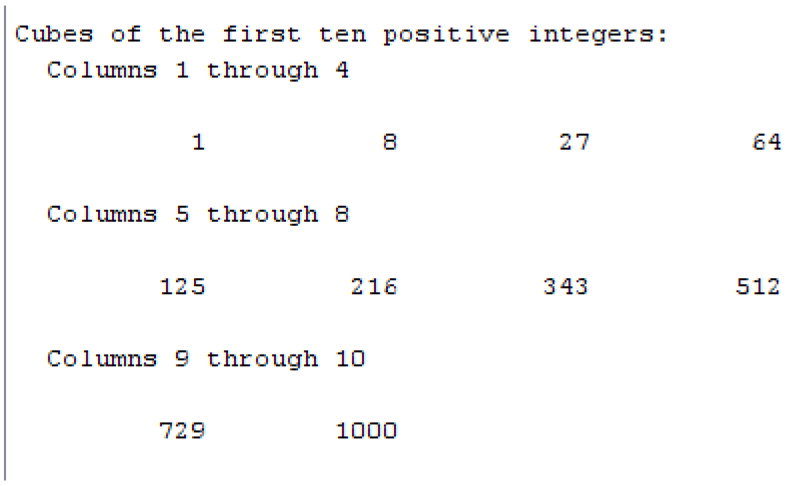
1. Initialize an empty array or list to store the cubes of the numbers.
2. Use a for loop to iterate through the values from 1 to 10.
3. Within each iteration:
4. Calculate the cube of the current number by multiplying it by itself twice.
5. Store the calculated cube in the array.
6. After the loop completes, display or return the array containing the cubes of the numbers.

Code



```
cubes = zeros(10, 1); % Preallocate an array to store the cubes

for i = 1:10
    cubes(i) = i^3; % Cube each positive integer from 1 to 10
end

disp('Cubes of the first ten positive integers:');
disp(cubes');
```

Output



```
Cubes of the first ten positive integers:
  Columns 1 through 4

           1           8          27          64

  Columns 5 through 8

         125         216         343         512

  Columns 9 through 10

         729        1000
```

Conclusion

The algorithm efficiently calculates and stores the cubes of the first ten positive integers using a for loop in MATLAB. The resulting array contains the cubes of the numbers from 1 to 10, providing an easy and organized way to access these values for further use or display.

TASK 07

Add the following two matrices using for loop.

$$A = \begin{bmatrix} 5 & 12 & 3 \\ 9 & 6 & 5 \\ 2 & 2 & 1 \end{bmatrix}, B = \begin{bmatrix} 2 & 1 & 9 \\ 10 & 5 & 6 \\ 3 & 4 & 2 \end{bmatrix}$$

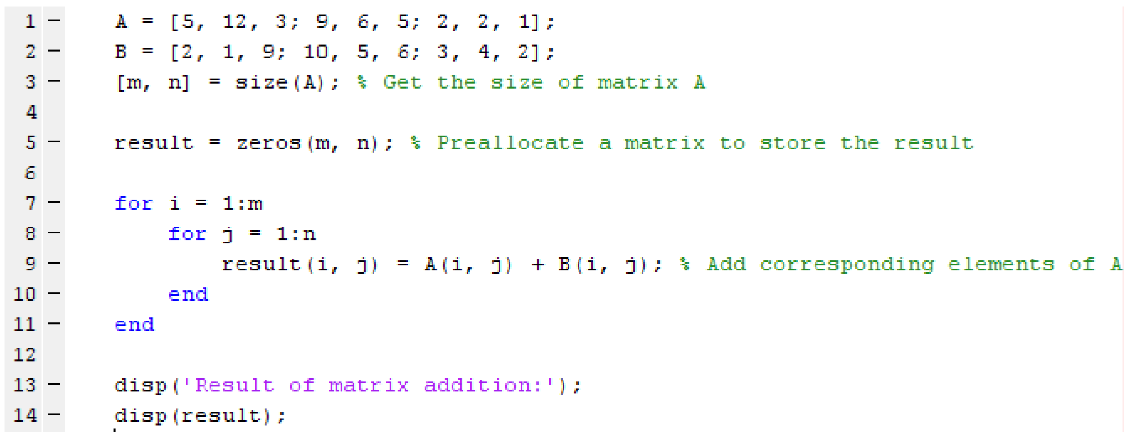
Problem Analysis

We are provided with a matrices and asked to perform addition operations on it using loops.

Algorithm

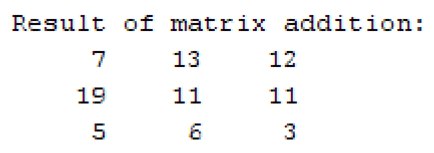
- Initialize an empty matrix C with the same size as matrices A and B to store the result.
- Use nested for loops to iterate through each element of matrices A and B.
- Add the corresponding elements from A and B and store the result in the corresponding element of matrix C.
- After completing the loop, matrix C will contain the sum of matrices A and B.

Code



```matlab
A = [5, 12, 3; 9, 6, 5; 2, 2, 1];
B = [2, 1, 9; 10, 5, 6; 3, 4, 2];
[m, n] = size(A); % Get the size of matrix A

result = zeros(m, n); % Preallocate a matrix to store the result

for i = 1:m
    for j = 1:n
        result(i, j) = A(i, j) + B(i, j); % Add corresponding elements of A and B
    end
end

disp('Result of matrix addition:');
disp(result);
```

Output



```
Result of matrix addition:
     7    13    12
    19    11    11
     5     6     3
```

Conclusion

It outlines the steps to perform the specified operations on the given matrices using loops.

TASK 09

Problem Analysis:

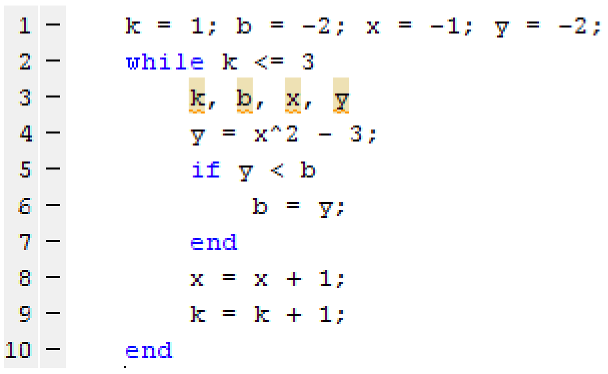
The problem involves a while loop that executes as long as k is less than or equal to 3. Within each iteration of the loop, the values of k, b, x, and y are displayed. The value of y is updated based on the formula y = x^2 - 3, and if y is less than b, b is updated to the value of y. Additionally, x is incremented by 1 in each iteration, and k is incremented by 1 at the end of each iteration.

Algorithm:

1. Initialize variables k = 1, b = -2, x = -1, and y = -2.
2. Enter a while loop that continues as long as k is less than or equal to 3.
3. Within each iteration of the loop:

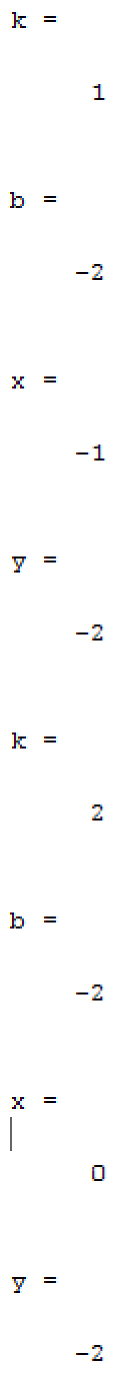
4. Display the values of k, b, x, and y.
5. Calculate the value of y using the formula y = x^2 - 3.
6. Check if y is less than b. If so, update b to the value of y.
7. Increment x by 1.
8. Increment k by 1.
9. Exit the loop when k exceeds 3.

Code:

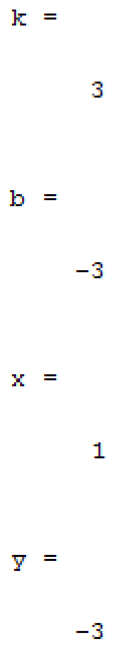


```
k = 1; b = -2; x = -1; y = -2;
while k <= 3
    k, b, x, y
    y = x^2 - 3;
    if y < b
        b = y;
    end
    x = x + 1;
    k = k + 1;
end
```

Output



```
k =

     1


b =

    -2


x =

    -1


y =

    -2


k =

     2


b =

    -2


x =

     0


y =

    -2
```



```
k =

     3


b =

    -3


x =

     1


y =

    -3
```

Conclusion

The script demonstrates the use of a while loop to perform iterative computations and updates based on specific conditions.

TASK 10

Create an M-file that inputs a number from user and then finds out the factorial of that number.

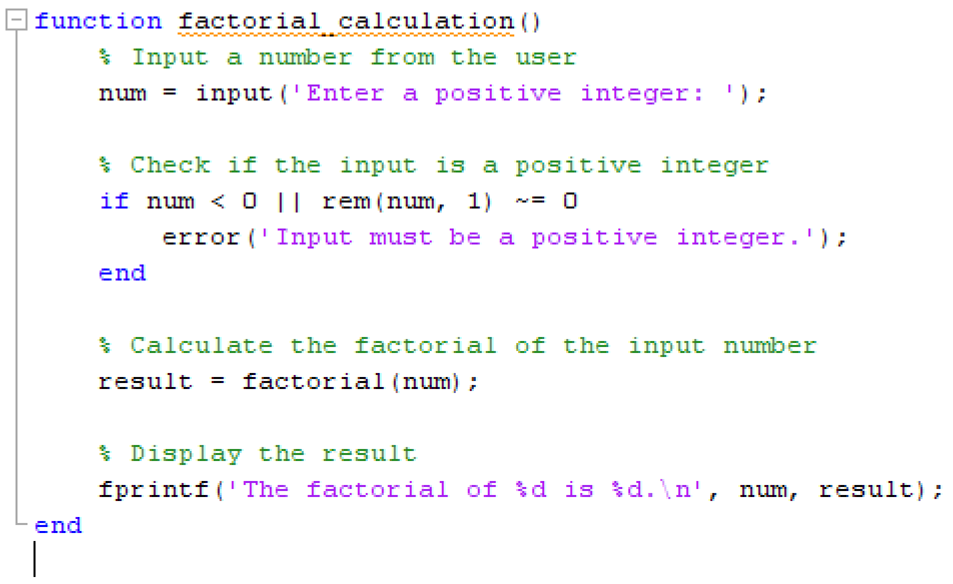
Problem Analysis

The function aims to calculate the factorial of a positive integer entered by the user.

Algorithm

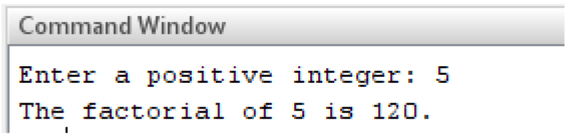
1. Prompt the user to enter a positive integer.
2. Check if the input is a positive integer. If not, display an error message.
3. Calculate the factorial of the input number using the built-in MATLAB function factorial().
4. Display the calculated factorial.

Code



```
function factorial_calculation()
    % Input a number from the user
    num = input('Enter a positive integer: ');

    % Check if the input is a positive integer
    if num < 0 || rem(num, 1) ~= 0
        error('Input must be a positive integer.');
    end

    % Calculate the factorial of the input number
    result = factorial(num);

    % Display the result
    fprintf('The factorial of %d is %d.\n', num, result);
end
```

Output



```
Enter a positive integer: 5
The factorial of 5 is 120.
```

Conclusion

This algorithm outlines the steps to find inverse of Linear equations using MATLAB.

TASK 11

Create an M-file that takes two vectors from user. Make sure that the second vector taken is of the same size as the first vector (Hint: use while loop). In a while loop, generate a third vector that contains the sum of the squares of corresponding entries of both the vectors.

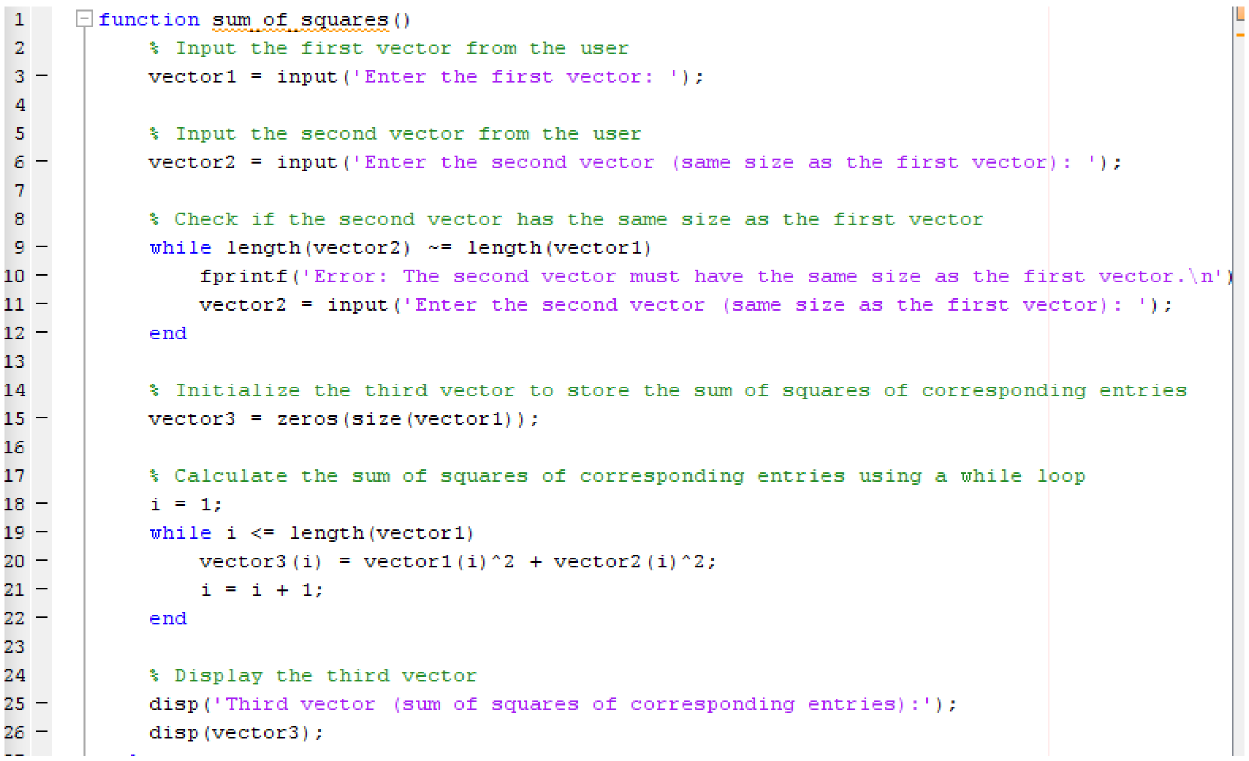
Problem Analysis

The problem involves taking two vectors from the user, ensuring that the second vector is of the same size as the first vector. Then, in a while loop, the program generates a third vector that contains the sum of the squares of corresponding entries of both vectors.

Algorithm

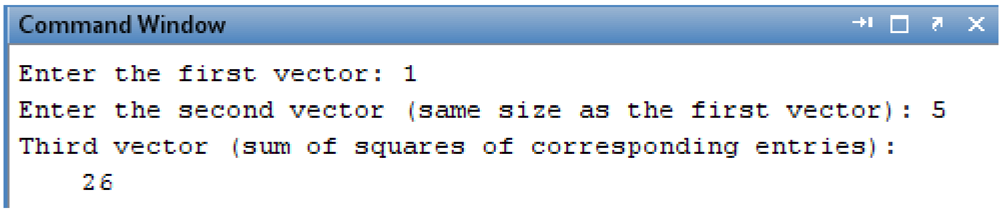
1. Prompt the user to enter the first vector (vector1) and ensure it is a valid numerical vector.
2. Prompt the user to enter the second vector (vector2) and ensure it is of the same size as vector1.
3. Initialize an empty vector (result_vector) to store the sum of squares of corresponding entries.
4. Use a while loop with an index i starting from 1 and incrementing until the end of vector1.
5. Calculate the square of the i-th element of vector1 and vector2.
6. Add the squares and store the result in result_vector[i].
7. Display the resulting vector result_vector.

Code



```
function sum_of_squares()
    % Input the first vector from the user
    vector1 = input('Enter the first vector: ');

    % Input the second vector from the user
    vector2 = input('Enter the second vector (same size as the first vector): ');

    % Check if the second vector has the same size as the first vector
    while length(vector2) ~= length(vector1)
        fprintf('Error: The second vector must have the same size as the first vector.\n')
        vector2 = input('Enter the second vector (same size as the first vector): ');
    end

    % Initialize the third vector to store the sum of squares of corresponding entries
    vector3 = zeros(size(vector1));

    % Calculate the sum of squares of corresponding entries using a while loop
    i = 1;
    while i <= length(vector1)
        vector3(i) = vector1(i)^2 + vector2(i)^2;
        i = i + 1;
    end

    % Display the third vector
    disp('Third vector (sum of squares of corresponding entries):');
    disp(vector3);
```

Output



Command Window

```
Enter the first vector: 1
Enter the second vector (same size as the first vector): 5
Third vector (sum of squares of corresponding entries):
    26
```

Conclusion

The use of a while loop ensures user input compliance and enhances the robustness of the program.

Task 12:

Perform the following commands on various matrices and comment on each.

```
>> ~A
>> A&B
>> A & ~B
>> A | B
>> A | ~A
```

Comments on the Commands:

~A: This command calculates the logical NOT of matrix A, flipping the logical values of all elements. It's useful for logical inversion or negation operations on a matrix.

A & B: This command performs element-wise logical AND operation between matrices A and B, resulting in a matrix with elements representing the AND operation between corresponding elements of A and B.

A & ~B: This command first negates matrix B and then performs element-wise logical AND operation between A and the negated B. It's useful for operations where certain elements need to be excluded based on conditions.

A | B: This command performs element-wise logical OR operation between matrices A and B, resulting in a matrix with elements representing the OR operation between corresponding elements of A and B.

A | ~A: This command first negates matrix A and then performs element-wise logical OR operation between A and its negation. It's a specific case where the result will always be a matrix of 1s, useful for certain logical evaluations or constructions.

Task 13:

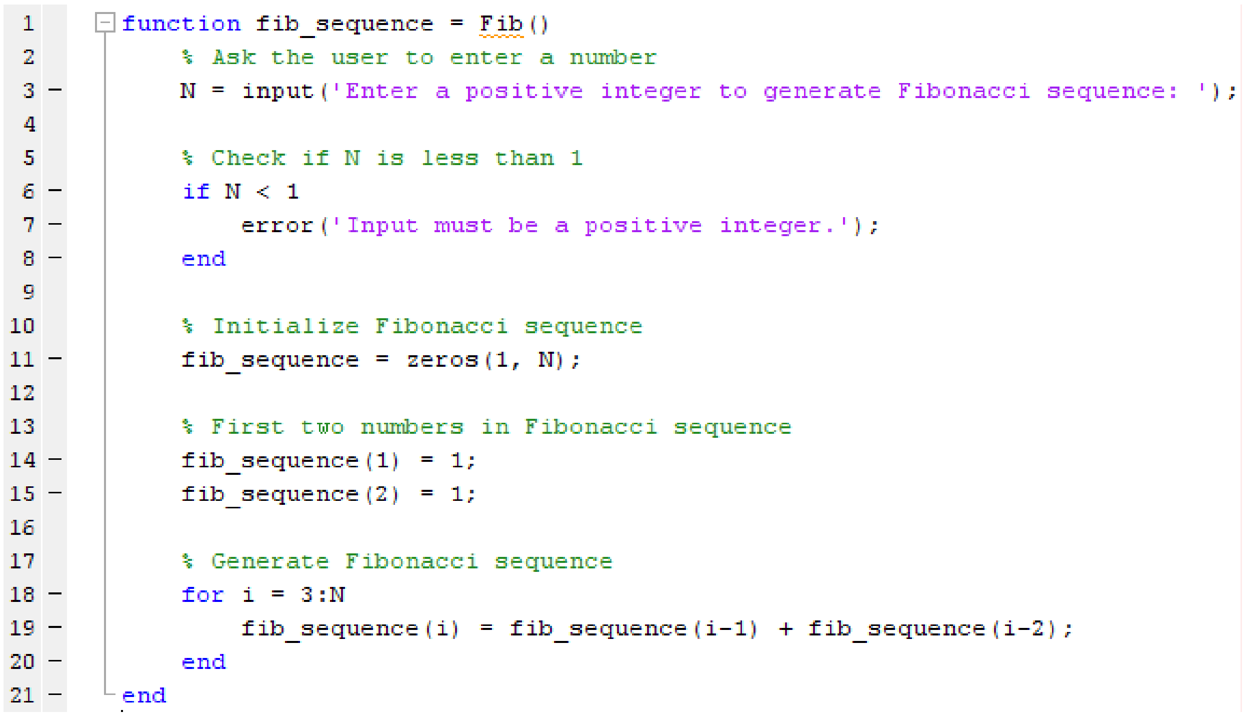
Problem Statement:

Design a function Fib(N) that takes N as an input and generates a Fibonacci sequence for N. Fibonacci sequence is a tile of squares whose side lengths are successive or each number is the sum of the previous number.

Problem Analysis:

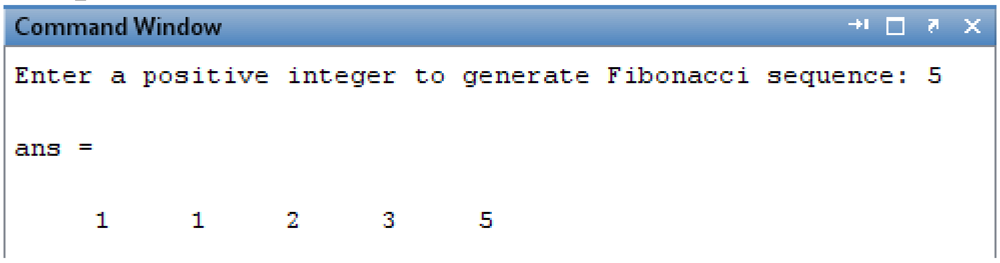
The problem involves creating a MATLAB function that prompts the user to enter a positive integer and then generates the corresponding Fibonacci sequence. The function should handle input validation to ensure that the user provides a valid positive integer.

Code:



```
function fib_sequence = Fib()
    % Ask the user to enter a number
    N = input('Enter a positive integer to generate Fibonacci sequence: ');

    % Check if N is less than 1
    if N < 1
        error('Input must be a positive integer.');
    end

    % Initialize Fibonacci sequence
    fib_sequence = zeros(1, N);

    % First two numbers in Fibonacci sequence
    fib_sequence(1) = 1;
    fib_sequence(2) = 1;

    % Generate Fibonacci sequence
    for i = 3:N
        fib_sequence(i) = fib_sequence(i-1) + fib_sequence(i-2);
    end
end
```

Output:



```
Enter a positive integer to generate Fibonacci sequence: 5

ans =

     1     1     2     3     5
```

Conclusion:

The function provides a straightforward way for users to interactively generate and view Fibonacci sequences in MATLAB.

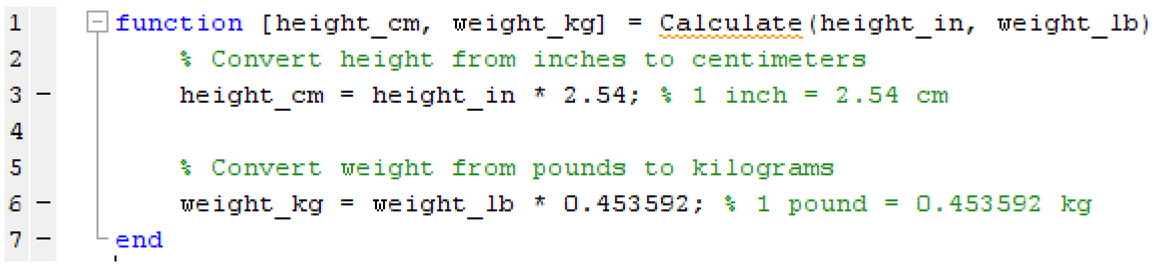
Task 14:

Write a user-defined MATLAB function Calculate, with two input and two output arguments that determines the height in centimeters (cm) and mass in kilograms (kg) of a person from his height in inches (in.) and weight in pounds (lb).

a) Determine the height and mass of a 5 ft. 15 in. person in SI units who weighs 180 lb.

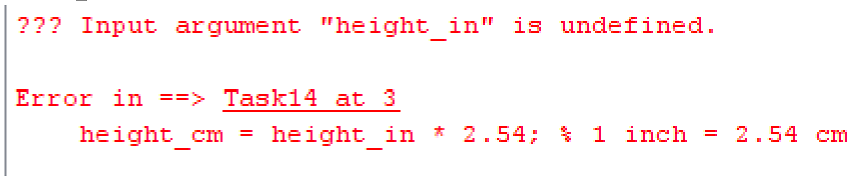
b) Determine your own height and weight in SI units.

Code:



```
function [height_cm, weight_kg] = Calculate(height_in, weight_lb)
    % Convert height from inches to centimeters
    height_cm = height_in * 2.54; % 1 inch = 2.54 cm

    % Convert weight from pounds to kilograms
    weight_kg = weight_lb * 0.453592; % 1 pound = 0.453592 kg
end
```

Output:



```
??? Input argument "height_in" is undefined.

Error in ==> Task14 at 3
    height_cm = height_in * 2.54; % 1 inch = 2.54 cm
```

Conclusion:

The Calculate function provides a convenient way to convert height from inches to centimeters and weight from pounds to kilograms in MATLAB.

Task 15:

File handling in MATLAB. Create files of different formats in MATLAB. Use the following commands to create a text file using MATLAB commands:

1. Open a file using fopen

op = fopen('weekdays.txt','wt');

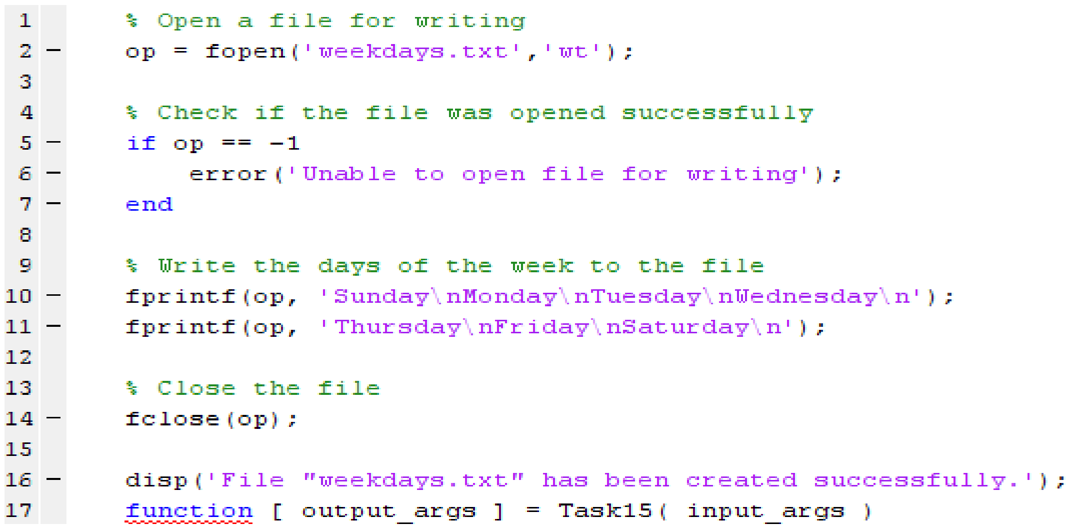
2. Write the output using fprintf

    fprintf(op,'Sunday\nMonday\nTuesday\nWednesday\n');
    fprintf(op,'Thursday\nFriday\nSaturday\n');

3. Close the file using fclose

    fclose(op);

Code:



```
% Open a file for writing
op = fopen('weekdays.txt','wt');

% Check if the file was opened successfully
if op == -1
    error('Unable to open file for writing');
end

% Write the days of the week to the file
fprintf(op, 'Sunday\nMonday\nTuesday\nWednesday\n');
fprintf(op, 'Thursday\nFriday\nSaturday\n');

% Close the file
fclose(op);

disp('File "weekdays.txt" has been created successfully.');
function [ output_args ] = Task15( input_args )
```

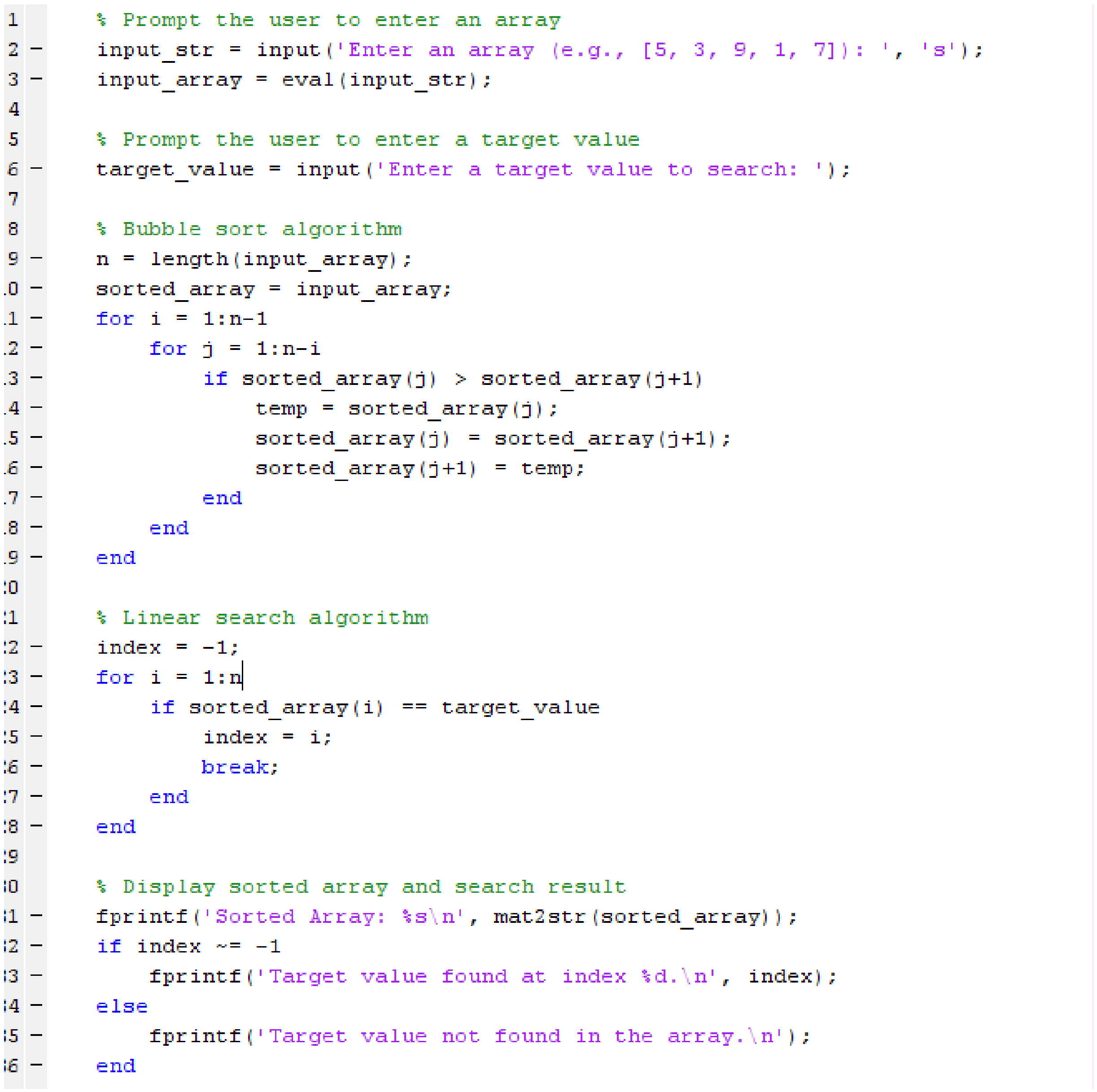
Task 16:

Implement any sorting and searching algorithm of your choice by creating user-defined functions in MATLAB.

Problem Statement:

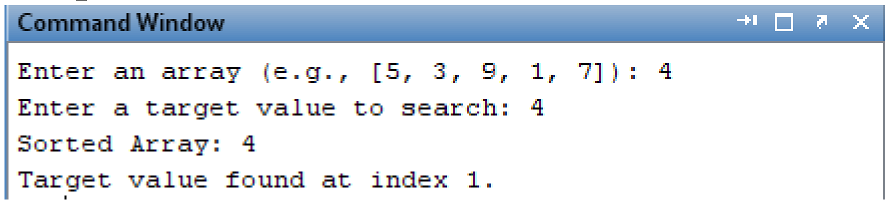
The MATLAB script prompts the user to input an array and a target value. It then performs sorting using the bubble sort algorithm and searches for the target value using linear search. Finally, it displays the sorted array and indicates whether the target value is found and at which index if applicable.

Code:



```matlab
% Prompt the user to enter an array
input_str = input('Enter an array (e.g., [5, 3, 9, 1, 7]): ', 's');
input_array = eval(input_str);

% Prompt the user to enter a target value
target_value = input('Enter a target value to search: ');

% Bubble sort algorithm
n = length(input_array);
sorted_array = input_array;
for i = 1:n-1
    for j = 1:n-i
        if sorted_array(j) > sorted_array(j+1)
            temp = sorted_array(j);
            sorted_array(j) = sorted_array(j+1);
            sorted_array(j+1) = temp;
        end
    end
end

% Linear search algorithm
index = -1;
for i = 1:n
    if sorted_array(i) == target_value
        index = i;
        break;
    end
end

% Display sorted array and search result
fprintf('Sorted Array: %s\n', mat2str(sorted_array));
if index ~= -1
    fprintf('Target value found at index %d.\n', index);
else
    fprintf('Target value not found in the array.\n');
end
```

Output:



```
Enter an array (e.g., [5, 3, 9, 1, 7]): 4
Enter a target value to search: 4
Sorted Array: 4
Target value found at index 1.
```