SET OF PROGRAMS

1.Statistical Analysis Tool
Function Prototype: void computeStats(const double *array, int size, double *average, double *variance)
Data Types: const double*, int, double*
Concepts: Pointers, arrays, functions, passing constant data, pass by reference.
Details: Compute the average and variance of an array of experimental results, ensuring the function uses pointers for accessing the data and modifying the results.

```
#include <stdio.h>

// Function to compute the average and variance
void computeStats(const double *array, int size, double *average, double *variance) {
    // Ensure that the array size is positive
    if (size <= 0) {
        *average = 0;
        *variance = 0;
        return;
    }

    double sum = 0;
    double sumOfSquares = 0;

    // Calculate the sum of the array and the sum of squares
    for (int i = 0; i < size; i++) {
        sum += array[i];
        sumOfSquares += array[i] * array[i];
    }

    // Compute the average
    *average = sum / size;

    // Compute the variance
    *variance = (sumOfSquares / size) - (*average * *average);
}
```

2.Data Normalization
Function Prototype: double* normalizeData(const double *array, int size)
Data Types: const double*, int, double*
Concepts: Arrays, functions returning pointers, loops.
Details: Normalize data points in an array, returning a pointer to the new normalized array.

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>  // For FLT_MAX and FLT_MIN

// Function to normalize data points in an array
double* normalizeData(const double *array, int size) {
    if (size <= 0) return NULL;  // Ensure that the array size is valid

    // Find the min and max values in the array
    double min = DBL_MAX, max = -DBL_MAX;
```

```
    for (int i = 0; i < size; i++) {
        if (array[i] < min) {
            min = array[i];
        }
        if (array[i] > max) {
            max = array[i];
        }
    }

    // If all values are the same, return an array of zeros (avoiding division by zero)
    if (min == max) {
        double *normalizedArray = (double *)malloc(size * sizeof(double));
        if (normalizedArray == NULL) {
            return NULL;  // Memory allocation failed
        }
        for (int i = 0; i < size; i++) {
            normalizedArray[i] = 0.0;  // All values normalized to zero
        }
        return normalizedArray;
    }

    // Allocate memory for the normalized array
    double *normalizedArray = (double *)malloc(size * sizeof(double));
    if (normalizedArray == NULL) {
        return NULL;  // Memory allocation failed
    }

    // Normalize each element in the array
    for (int i = 0; i < size; i++) {
        normalizedArray[i] = (array[i] - min) / (max - min);
    }

    return normalizedArray;  // Return pointer to the new array
}

// Helper function to print an array for testing
void printArray(const double *array, int size) {
    for (int i = 0; i < size; i++) {
        printf("%f ", array[i]);
    }
    printf("\n");
}

int main() {
    double data[] = {5.0, 10.0, 15.0, 20.0, 25.0};
    int size = sizeof(data) / sizeof(data[0]);

    // Normalize the data
    double *normalizedData = normalizeData(data, size);

    // Print the normalized data
    if (normalizedData != NULL) {
        printArray(normalizedData, size);

        // Free the allocated memory for the normalized array
```

```
        free(normalizedData);
    }

    return 0;
}
```

3.Experimental Report Generator
Function Prototype: void generateReport(const double *results, const char *descriptions[], int size)
Data Types: const double*, const char*[], int
Concepts: Strings, arrays, functions, passing constant data.
Details: Generate a report summarizing experimental results and their descriptions, using constant data to ensure the input is not modified.

```
#include <stdio.h>

void generateReport(const double *results, const char *descriptions[], int size) {
    // Check if the size is greater than 0 to prevent empty input arrays
    if (size <= 0) {
        printf("No results to report.\n");
        return;
    }

    printf("Experimental Results Report:\n");
    printf("----------------------------\n");

    // Iterate through the arrays and print the results and descriptions
    for (int i = 0; i < size; i++) {
        printf("Experiment %d:\n", i + 1);
        printf("Description: %s\n", descriptions[i]);
        printf("Result: %.2f\n", results[i]);
        printf("----------------------------\n");
    }
}

int main() {
    // Example data for the report
    double results[] = {23.5, 47.8, 12.9, 88.6};
    const char *descriptions[] = {
        "Test of material strength",
        "Measurement of temperature fluctuation",
        "Evaluation of fluid viscosity",
        "Analysis of electrical resistance"
    };
    int size = sizeof(results) / sizeof(results[0]);  // Determine the number of results

    // Call the function to generate the report
    generateReport(results, descriptions, size);

    return 0;
}
```

4.Data Anomaly Detector
Function Prototype: void detectAnomalies(const double *data, int size, double threshold, int *anomalyCount)
Data Types: const double*, int, double, int*

Concepts: Decision-making, arrays, pointers, functions.
Details: Detect anomalies in a dataset based on a threshold, updating the anomaly count by reference.

```
#include <stdio.h>

void detectAnomalies(const double *data, int size, double threshold, int *anomalyCount) {
    // Initialize the anomaly count to 0
    *anomalyCount = 0;

    // Loop through the array to check each value
    for (int i = 0; i < size; i++) {
        if (data[i] > threshold) {
            (*anomalyCount)++;  // Increment the anomaly count if the value exceeds the threshold
        }
    }
}

int main() {
    // Example dataset
    double data[] = {1.2, 3.5, 6.7, 2.8, 9.1, 3.4};
    int size = sizeof(data) / sizeof(data[0]);
    double threshold = 5.0;
    int anomalyCount;

    // Call the function to detect anomalies
    detectAnomalies(data, size, threshold, &anomalyCount);

    // Output the result
    printf("Number of anomalies: %d\n", anomalyCount);

    return 0;
}
```

5.Data Classifier
Function Prototype: void classifyData(const double *data, int size, char *labels[], double threshold)
Data Types: const double*, int, char*[], double
Concepts: Decision-making, arrays, functions, pointers.
Details: Classify data points into categories based on a threshold, updating an array of labels.

```
#include <stdio.h>

void classifyData(const double *data, int size, char *labels[], double threshold) {
    // Loop through each data point
    for (int i = 0; i < size; i++) {
        // Compare the data point with the threshold
        if (data[i] >= threshold) {
            labels[i] = "High";  // Label as High if data[i] is greater than or equal to threshold
        } else {
            labels[i] = "Low";   // Label as Low if data[i] is less than threshold
        }
    }
}

int main() {
    // Example data points and size
```

```
    double data[] = {10.5, 3.2, 8.7, 14.0, 6.3};
    int size = sizeof(data) / sizeof(data[0]);

    // Array to store labels
    char *labels[size];

    // Define a threshold
    double threshold = 7.0;

    // Call classifyData function
    classifyData(data, size, labels, threshold);

    // Print the classified data with labels
    for (int i = 0; i < size; i++) {
        printf("Data: %.2f, Label: %s\n", data[i], labels[i]);
    }

    return 0;
}
```

6.Neural Network Weight Adjuster
Function Prototype: void adjustWeights(double *weights, int size, double learningRate)
Data Types: double*, int, double
Concepts: Pointers, arrays, functions, loops.
Details: Adjust neural network weights using a given learning rate, with weights passed by reference.

```
#include <stdio.h>

// Function to adjust the weights based on the learning rate
void adjustWeights(double *weights, int size, double learningRate) {
    // Iterate over all the weights
    for (int i = 0; i < size; i++) {
        // Update weight (simplified gradient descent step)
        weights[i] -= learningRate * 1;  // Assuming gradient is 1
    }
}

int main() {
    // Example usage
    double weights[] = {0.5, -0.2, 0.8};  // Example weights
    int size = 3;  // Number of weights
    double learningRate = 0.1;  // Example learning rate

    printf("Weights before adjustment:\n");
    for (int i = 0; i < size; i++) {
        printf("Weight %d: %.2f\n", i, weights[i]);
    }

    // Call the function to adjust the weights
    adjustWeights(weights, size, learningRate);

    printf("\nWeights after adjustment:\n");
    for (int i = 0; i < size; i++) {
        printf("Weight %d: %.2f\n", i, weights[i]);
    }
```

```
    return 0;
}
```

7.AI Model Evaluator
Function Prototype: void evaluateModels(const double *accuracies, int size, double *bestAccuracy)
Data Types: const double*, int, double*
Concepts: Loops, arrays, functions, pointers.
Details: Evaluate multiple AI models, determining the best accuracy and updating it by reference.

```
#include <stdio.h>

void evaluateModels(const double *accuracies, int size, double *bestAccuracy) {
    // Check if the size is valid
    if (size <= 0) {
        *bestAccuracy = 0.0;  // No models, set best accuracy to 0
        return;
    }

    // Initialize bestAccuracy to the first model's accuracy
    *bestAccuracy = accuracies[0];

    // Loop through the accuracies to find the best one
    for (int i = 1; i < size; i++) {
        if (accuracies[i] > *bestAccuracy) {
            *bestAccuracy = accuracies[i];  // Update best accuracy if a better one is found
        }
    }
}

int main() {
    // Example of using the evaluateModels function
    double accuracies[] = {0.85, 0.92, 0.78, 0.95, 0.88};
    int size = sizeof(accuracies) / sizeof(accuracies[0]);
    double bestAccuracy;

    evaluateModels(accuracies, size, &bestAccuracy);

    printf("The best accuracy is: %.2f\n", bestAccuracy);

    return 0;
}
```

8.Decision Tree Constructor
Function Prototype: void constructDecisionTree(const double *features, int size, int *treeStructure)
Data Types: const double*, int, int*
Concepts: Decision-making, arrays, functions.
Details: Construct a decision tree based on feature data, updating the tree structure by reference

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

#define MAX_TREE_NODES 100

// Function to initialize a tree node's decision in arrays
```

```
void initTreeNode(int *featureIndex, double *threshold, int *leftChild, int *rightChild, int nodeIndex, int
featureIdx, double thresholdVal, int left, int right) {
    featureIndex[nodeIndex] = featureIdx;
    threshold[nodeIndex] = thresholdVal;
    leftChild[nodeIndex] = left;
    rightChild[nodeIndex] = right;
}

// Function to construct a decision tree
void constructDecisionTree(const double *features, int size, int *treeStructure) {
    // Arrays to represent the tree structure
    int featureIndex[MAX_TREE_NODES];  // Store the feature index used in each node
    double threshold[MAX_TREE_NODES];   // Store the threshold used in each node
    int leftChild[MAX_TREE_NODES];     // Left child index for each node
    int rightChild[MAX_TREE_NODES];    // Right child index for each node

    int nodeIndex = 0;  // Start with the root node
    double thresholdVal = 0.5;  // Example threshold value

    // Initialize the root node with the first feature and threshold
    initTreeNode(featureIndex, threshold, leftChild, rightChild, nodeIndex, 0, thresholdVal, -1, -1);

    // The tree root is at index 0
    treeStructure[0] = nodeIndex;

    // Simple decision-making (just an example): split based on the first feature and threshold
    for (int i = 0; i < size; i++) {
        if (features[i] <= thresholdVal) {
            printf("Feature %d <= %.2f, go left\n", i, thresholdVal);
        } else {
            printf("Feature %d > %.2f, go right\n", i, thresholdVal);
        }
    }

    // Example: You could add left and right children if you wanted to extend the tree
    // But here we just keep it simple with no further branching for now
}

int main() {
    // Example feature data
    double features[] = {0.3, 0.8, 0.2, 0.7, 0.6};
    int size = 5;  // Number of features
    int treeStructure[MAX_TREE_NODES];  // Stores the structure of the tree

    // Construct the decision tree
    constructDecisionTree(features, size, treeStructure);

    return 0;
}
```

9.Sentiment Analysis Processor
Function Prototype: void processSentiments(const char *sentences[], int size, int *sentimentScores)
Data Types: const char*[], int, int*
Concepts: Strings, arrays, functions, pointers.
Details: Analyze sentiments of sentences, updating sentiment scores by reference.

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

void processSentiments(const char *sentences[], int size, int *sentimentScores) {
    // Define simple positive and negative words
    const char *positiveWords[] = {"good", "happy", "great"};
    const char *negativeWords[] = {"bad", "sad", "angry"};
    int numPositive = sizeof(positiveWords) / sizeof(positiveWords[0]);
    int numNegative = sizeof(negativeWords) / sizeof(negativeWords[0]);

    // Loop through each sentence
    for (int i = 0; i < size; i++) {
        int score = 0;
        const char *sentence = sentences[i];

        // Check for positive words
        for (int j = 0; j < numPositive; j++) {
            if (strstr(sentence, positiveWords[j]) != NULL) {
                score++;
            }
        }

        // Check for negative words
        for (int j = 0; j < numNegative; j++) {
            if (strstr(sentence, negativeWords[j]) != NULL) {
                score--;
            }
        }

        // Update sentimentScores array
        sentimentScores[i] = score;
    }
}

int main() {
    const char *sentences[] = {
        "I am feeling good today",
        "This is a bad day",
        "I am so happy and great",
        "I am sad and angry"
    };

    int size = sizeof(sentences) / sizeof(sentences[0]);
    int sentimentScores[size];

    // Process sentiments
    processSentiments(sentences, size, sentimentScores);

    // Print the results
    for (int i = 0; i < size; i++) {
        printf("Sentence: %s\n", sentences[i]);
        printf("Sentiment Score: %d\n\n", sentimentScores[i]);
    }
```

```
    return 0;
}
```

10.Training Data Generator
Function Prototype: double* generateTrainingData(const double *baseData, int size, int multiplier)
Data Types: const double*, int, double*
Concepts: Arrays, functions returning pointers, loops.
Details: Generate training data by applying a multiplier to base data, returning a pointer to the new data array.

```
#include <stdio.h>

void generateTrainingData(const double *baseData, double *newData, int size, int multiplier) {
    // Loop through the base data and apply the multiplier
    for (int i = 0; i < size; i++) {
        newData[i] = baseData[i] * multiplier;
    }
}

int main() {
    // Sample base data
    double baseData[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    int size = 5;
    int multiplier = 2;

    // Create an array to store the new data
    double newData[size];

    // Generate the training data
    generateTrainingData(baseData, newData, size, multiplier);

    // Print the generated training data
    printf("Generated training data:\n");
    for (int i = 0; i < size; i++) {
        printf("%lf ", newData[i]);
    }
    printf("\n");

    return 0;
}
```

11.Image Filter Application
Function Prototype: void applyFilter(const unsigned char *image, unsigned char *filteredImage, int width, int height)
Data Types: const unsigned char*, unsigned char*, int
Concepts: Arrays, pointers, functions.
Details: Apply a filter to an image, modifying the filtered image by reference.

```
#include <stdio.h>

void applyFilter(const unsigned char *image, unsigned char *filteredImage, int width, int height) {
    int totalPixels = width * height;

    // Iterate over each pixel of the image
    for (int i = 0; i < totalPixels; i++) {
```

```
        // Calculate the index of the pixel's red, green, and blue components in the image array
        int rIdx = i * 3;
        int gIdx = rIdx + 1;
        int bIdx = rIdx + 2;

        // Get the RGB values from the original image
        unsigned char r = image[rIdx];
        unsigned char g = image[gIdx];
        unsigned char b = image[bIdx];

        // Calculate the grayscale value (average of R, G, and B)
        unsigned char gray = (unsigned char)((r + g + b) / 3);

        // Set the filtered image to the grayscale value for R, G, and B
        filteredImage[rIdx] = gray;
        filteredImage[gIdx] = gray;
        filteredImage[bIdx] = gray;
    }
}

int main() {
    // Example image (3x3 pixels, RGB format)
    unsigned char image[27] = {
        255, 0, 0,   // Red
        0, 255, 0,   // Green
        0, 0, 255,   // Blue
        255, 255, 0, // Yellow
        0, 255, 255, // Cyan
        255, 0, 255, // Magenta
        192, 192, 192, // Light gray
        128, 128, 128, // Gray
        0, 0, 0      // Black
    };

    // Create an array to store the filtered image
    unsigned char filteredImage[27];

    // Apply the filter to the image (3x3 pixels)
    applyFilter(image, filteredImage, 3, 3);

    // Print the filtered image (grayscale)
    printf("Filtered Image (Grayscale):\n");
    for (int i = 0; i < 27; i += 3) {
        printf("R: %d, G: %d, B: %d\n", filteredImage[i], filteredImage[i+1], filteredImage[i+2]);
    }

    return 0;
}
```

12.Edge Detection Algorithm
Function Prototype: void detectEdges(const unsigned char *image, unsigned char *edges, int width, int height)
Data Types: const unsigned char*, unsigned char*, int
Concepts: Loops, arrays, decision-making, functions.
Details: Detect edges in an image, updating the edges array by reference.

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

// Example grayscale image (3x3 for simplicity)
int main() {
    unsigned char image[9] = { 0, 0, 0,
                               255, 255, 255,
                               0, 0, 0 };

    unsigned char edges[9];  // To store the edge-detected result

    // Call the edge detection function
    detectEdges(image, edges, 3, 3);

    // Print the edges result
    for (int i = 0; i < 9; i++) {
        printf("%d ", edges[i]);
        if ((i + 1) % 3 == 0) printf("\n");
    }

    return 0;
}


void detectEdges(const unsigned char *image, unsigned char *edges, int width, int height) {
    // Sobel kernels for horizontal (Gx) and vertical (Gy) gradients
    int Gx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
    int Gy[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};

    // Loop through each pixel, excluding the borders (to avoid accessing out of bounds)
    for (int y = 1; y < height - 1; y++) {
        for (int x = 1; x < width - 1; x++) {
            int sumX = 0;
            int sumY = 0;

            // Apply Sobel filter (kernel convolution)
            for (int ky = -1; ky <= 1; ky++) {
                for (int kx = -1; kx <= 1; kx++) {
                    int pixel = image[(y + ky) * width + (x + kx)];
                    sumX += pixel * Gx[ky + 1][kx + 1];
                    sumY += pixel * Gy[ky + 1][kx + 1];
                }
            }

            // Calculate the gradient magnitude
            int magnitude = (int) sqrt(sumX * sumX + sumY * sumY);

            // Set the edge pixel value in the edges array (clamp to 255 if needed)
            if (magnitude > 255) {
                edges[y * width + x] = 255;
            } else {
                edges[y * width + x] = (unsigned char) magnitude;
            }
```

```
        }
    }
}
```

13.Object Recognition System
Function Prototype: void recognizeObjects(const double *features, int size, char *objectLabels[])
Data Types: const double*, int, char*[]
Concepts: Decision-making, arrays, functions, pointers.
Details: Recognize objects based on feature vectors, updating an array of object labels.

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

// Define maximum number of objects and features for this example
#define MAX_OBJECTS 5
#define MAX_FEATURES 3

// Predefined feature vectors for objects (e.g., characteristics of the objects)
const double referenceFeatures[MAX_OBJECTS][MAX_FEATURES] = {
    {1.0, 2.0, 3.0},   // Features for Object 1
    {4.0, 5.0, 6.0},   // Features for Object 2
    {7.0, 8.0, 9.0},   // Features for Object 3
    {10.0, 11.0, 12.0}, // Features for Object 4
    {13.0, 14.0, 15.0}  // Features for Object 5
};

// Labels for the objects
const char *labels[MAX_OBJECTS] = {
    "Object 1",
    "Object 2",
    "Object 3",
    "Object 4",
    "Object 5"
};

// Function to calculate the Euclidean distance between two feature vectors
double calculateDistance(const double *features1, const double *features2, int size) {
    double sum = 0.0;
    for (int i = 0; i < size; i++) {
        sum += pow(features1[i] - features2[i], 2);
    }
    return sqrt(sum);
}

// Function to recognize objects based on the feature vector and return the label
void recognizeObjects(const double *features, int size, char *recognizedLabel) {
    double minDistance = INFINITY; // Start with a very high value
    int recognizedIndex = -1;

    // Compare the input features with each reference feature vector
    for (int i = 0; i < MAX_OBJECTS; i++) {
        double distance = calculateDistance(features, referenceFeatures[i], size);

        // If the distance is smaller, update the recognized object
```

```
        if (distance < minDistance) {
            minDistance = distance;
            recognizedIndex = i;
        }
    }

    // Set the recognized label based on the closest reference
    if (recognizedIndex != -1) {
        strcpy(recognizedLabel, labels[recognizedIndex]);
    } else {
        strcpy(recognizedLabel, "Unknown Object");
    }
}

int main() {
    // Example input feature vector (representing an object to recognize)
    double inputFeatures[MAX_FEATURES] = {4.1, 5.0, 6.1};

    // Array to store the recognized label
    char recognizedLabel[50];

    // Call the recognizeObjects function
    recognizeObjects(inputFeatures, MAX_FEATURES, recognizedLabel);

    // Output the recognized label
    printf("Recognized Object: %s\n", recognizedLabel);

    return 0;
}
```

14.Image Resizing Function
Function Prototype: void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth, int originalHeight, int newWidth, int newHeight)
Data Types: const unsigned char*, unsigned char*, int
Concepts: Arrays, functions, pointers.
Details: Resize an image to new dimensions, modifying the output image by reference.

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

// Function to resize the image
void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth, int originalHeight, int newWidth, int newHeight) {
    // Iterate over each pixel in the output image
    for (int y = 0; y < newHeight; y++) {
        for (int x = 0; x < newWidth; x++) {
            // Calculate corresponding pixel in the original image (using nearest neighbor)
            int originalX = (x * originalWidth) / newWidth;
            int originalY = (y * originalHeight) / newHeight;

            // Calculate index in the input and output image arrays (assuming grayscale image)
            int inputIndex = originalY * originalWidth + originalX;
            int outputIndex = y * newWidth + x;

            // Assign the pixel value from inputImage to outputImage (assuming grayscale)
```

```
            outputImage[outputIndex] = inputImage[inputIndex];
        }
    }
}

// Helper function to print an image (for debugging purposes)
void printImage(unsigned char *image, int width, int height) {
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            printf("%d ", image[y * width + x]);
        }
        printf("\n");
    }
}

int main() {
    // Example usage of resizeImage
    int originalWidth = 4, originalHeight = 4;
    int newWidth = 2, newHeight = 2;

    // Example input image (grayscale)
    unsigned char inputImage[16] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
    unsigned char outputImage[4]; // Resized image (2x2)

    // Resize the image
    resizeImage(inputImage, outputImage, originalWidth, originalHeight, newWidth, newHeight);

    // Print the resized image
    printImage(outputImage, newWidth, newHeight);

    return 0;
}
```

15.Color Balance Adjuster
Function Prototype: void balanceColors(const unsigned char *image, unsigned char *balancedImage, int width, int height)
Data Types: const unsigned char*, unsigned char*, int
Concepts: Arrays, functions, pointers, loops.
Details: Adjust the color balance of an image, updating the balanced image by reference.

```
#include <stdio.h>

// Function to adjust color balance of an image
void balanceColors(const unsigned char *image, unsigned char *balancedImage, int width, int height) {
    // Define scaling factors for R, G, B (this can be adjusted for desired effect)
    float redScale = 1.1;   // Increase red by 10%
    float greenScale = 1.0; // No change to green
    float blueScale = 0.9;  // Decrease blue by 10%

    // Iterate over each pixel
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int pixelIndex = (y * width + x) * 3; // Calculate index for the pixel (RGB components)

            unsigned char r = image[pixelIndex];       // Red channel
```

```
            unsigned char g = image[pixelIndex + 1];   // Green channel
            unsigned char b = image[pixelIndex + 2];   // Blue channel

            // Apply color balance by scaling the RGB values
            int newR = (int)(r * redScale);
            int newG = (int)(g * greenScale);
            int newB = (int)(b * blueScale);

            // Clamp the values to ensure they remain within the valid range [0, 255]
            if (newR > 255) newR = 255;
            if (newR < 0) newR = 0;
            if (newG > 255) newG = 255;
            if (newG < 0) newG = 0;
            if (newB > 255) newB = 255;
            if (newB < 0) newB = 0;

            // Store the adjusted color back into the balanced image
            balancedImage[pixelIndex] = (unsigned char)newR;
            balancedImage[pixelIndex + 1] = (unsigned char)newG;
            balancedImage[pixelIndex + 2] = (unsigned char)newB;
        }
    }
}

int main() {
    // Example usage with a small image (2x2 pixels, each with RGB values)
    int width = 2, height = 2;

    // Sample image with 2x2 pixels
    unsigned char image[12] = {
        255, 0, 0,     // Red pixel
        0, 255, 0,     // Green pixel
        0, 0, 255,     // Blue pixel
        255, 255, 0    // Yellow pixel (Red + Green)
    };

    unsigned char balancedImage[12]; // To store the adjusted image

    // Call the balanceColors function
    balanceColors(image, balancedImage, width, height);

    // Print out the balanced image data (RGB values)
    for (int i = 0; i < width * height * 3; i += 3) {
        printf("Pixel %d: R=%d, G=%d, B=%d\n", i / 3,
               balancedImage[i], balancedImage[i + 1], balancedImage[i + 2]);
    }

    return 0;
}
```

16.Pattern Recognition Algorithm
Function Prototype: void recognizePatterns(const char *patterns[], int size, int *matchCounts)
Data Types: const char*[], int, int*
Concepts: Strings, arrays, decision-making, pointers.
Details: Recognize patterns in a dataset, updating match counts by reference.

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

void recognizePatterns(const char *patterns[], int size, int *matchCounts) {
    // Initialize matchCounts to 0
    for (int i = 0; i < size; i++) {
        matchCounts[i] = 0; // Reset match count for each pattern
    }

    // Compare each pattern with every other pattern
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (strcmp(patterns[i], patterns[j]) == 0) {
                matchCounts[i]++; // Increment match count if patterns match
            }
        }
    }
}

int main() {
    // Test case
    const char *patterns[] = {
        "apple",
        "banana",
        "apple",
        "cherry",
        "banana",
        "apple"
    };

    int size = 6; // Number of patterns
    int matchCounts[size]; // Array to store match counts

    // Call the function
    recognizePatterns(patterns, size, matchCounts);

    // Print the match counts
    for (int i = 0; i < size; i++) {
        printf("Pattern \"%s\" matched %d times.\n", patterns[i], matchCounts[i]);
    }

    return 0;
}
```

17.Climate Data Analyzer
Function Prototype: void analyzeClimateData(const double *temperatureReadings, int size, double *minTemp, double *maxTemp)
Data Types: const double*, int, double*
Concepts: Decision-making, arrays, functions.
Details: Analyze climate data to find minimum and maximum temperatures, updating these values by reference.

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

```
void analyzeClimateData(const double *temperatureReadings, int size, double *minTemp, double
*maxTemp) {
    // Initialize minTemp and maxTemp with the first reading in the array
    *minTemp = temperatureReadings[0];
    *maxTemp = temperatureReadings[0];

    // Loop through the temperature readings to find the min and max
    for (int i = 1; i < size; i++) {
        if (temperatureReadings[i] < *minTemp) {
            *minTemp = temperatureReadings[i];  // Update minTemp if current value is lower
        }
        if (temperatureReadings[i] > *maxTemp) {
            *maxTemp = temperatureReadings[i];  // Update maxTemp if current value is higher
        }
    }
}

int main() {
    // Example temperature readings array
    double temperatureReadings[] = {25.5, 30.1, 22.8, 28.4, 35.0, 24.3};
    int size = sizeof(temperatureReadings) / sizeof(temperatureReadings[0]);

    double minTemp, maxTemp;

    // Call the function to analyze the data
    analyzeClimateData(temperatureReadings, size, &minTemp, &maxTemp);

    // Print the results
    printf("Minimum Temperature: %.2f\n", minTemp);
    printf("Maximum Temperature: %.2f\n", maxTemp);

    return 0;
}
```

18.Quantum Data Processor
Function Prototype: void processQuantumData(const double *measurements, int size, double
*processedData)
Data Types: const double*, int, double*
Concepts: Arrays, functions, pointers, loops.
Details: Process quantum measurement data, updating the processed data array by reference.

```
#include <stdio.h>

// Function prototype
void processQuantumData(const double *measurements, int size, double *processedData);

int main() {
    // Example quantum measurements
    double measurements[] = {0.5, 1.2, 3.4, 2.1, 0.7};
    int size = sizeof(measurements) / sizeof(measurements[0]);

    // Array to store processed data
    double processedData[size];

    // Process the quantum data
```

```
    processQuantumData(measurements, size, processedData);

    // Print processed data
    printf("Processed Data:\n");
    for (int i = 0; i < size; i++) {
        printf("%f ", processedData[i]);
    }
    printf("\n");

    return 0;
}

// Function to process the quantum measurement data
void processQuantumData(const double *measurements, int size, double *processedData) {
    // Example processing: scale each measurement by a constant factor (e.g., 2.0)
    const double scaleFactor = 2.0;

    for (int i = 0; i < size; i++) {
        // Process each measurement (e.g., multiply by scale factor)
        processedData[i] = measurements[i] * scaleFactor;
    }
}
```

19.Scientific Data Visualization
Function Prototype: void visualizeData(const double *data, int size, const char *title)
Data Types: const double*, int, const char*
Concepts: Arrays, functions, strings.
Details: Visualize scientific data with a given title, using constant data for the title.

```
#include <stdio.h>

void visualizeData(const double *data, int size, const char *title) {
    // Print the title
    printf("Title: %s\n", title);

    // Print the data
    printf("Data:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2f ", data[i]);  // Print each data point with 2 decimal places
    }
    printf("\n");
}

int main() {
    // Example data
    double data[] = {1.1, 2.2, 3.3, 4.4, 5.5};
    int size = 5;
    const char *title = "Scientific Data Visualization";

    // Visualize the data
    visualizeData(data, size, title);

    return 0;
}
```

20.Genetic Data Simulator
Function Prototype: double* simulateGeneticData(const double *initialData, int size, double mutationRate)
Data Types: const double*, int, double
Concepts: Arrays, functions returning pointers, loops.
Details: Simulate genetic data evolution by applying a mutation rate, returning a pointer to the simulated data.

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

double* simulateGeneticData(const double *initialData, int size, double mutationRate) {
    // Allocate memory for the new simulated data
    double *simulatedData = (double *)malloc(size * sizeof(double));

    if (simulatedData == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }

    // Seed the random number generator for mutation simulation
    srand(time(NULL));

    // Iterate through the initial genetic data and simulate mutations
    for (int i = 0; i < size; i++) {
        simulatedData[i] = initialData[i];

        // Check if mutation occurs based on mutationRate
        if ((rand() / (double)RAND_MAX) < mutationRate) {
            // Apply a small mutation, e.g., random value between -0.1 and 0.1
            simulatedData[i] += ((rand() / (double)RAND_MAX) * 0.2 - 0.1);  // Mutation range [-0.1, 0.1]
        }
    }

    return simulatedData;
}

int main() {
    // Example usage of the simulateGeneticData function

    int size = 5;
    double initialData[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    double mutationRate = 0.3;

    // Call the function
    double *simulatedData = simulateGeneticData(initialData, size, mutationRate);

    // Print the simulated data
    printf("Simulated Genetic Data:\n");
    for (int i = 0; i < size; i++) {
        printf("Data[%d] = %f\n", i, simulatedData[i]);
    }

    // Free the allocated memory
```

```
    free(simulatedData);

    return 0;
}
```

21.AI Performance Tracker
Function Prototype: void trackPerformance(const double *performanceData, int size, double *maxPerformance, double *minPerformance)
Data Types: const double*, int, double*
Concepts: Arrays, functions, pointers.
Details: Track AI performance data, updating maximum and minimum performance by reference.

```
#include <stdio.h>

void trackPerformance(const double *performanceData, int size, double *maxPerformance, double *minPerformance) {
    if (size <= 0) {
        return; // No data to process
    }

    // Initialize max and min with the first element of the performanceData array
    *maxPerformance = performanceData[0];
    *minPerformance = performanceData[0];

    // Iterate through the array to find the max and min
    for (int i = 1; i < size; i++) {
        if (performanceData[i] > *maxPerformance) {
            *maxPerformance = performanceData[i];
        }
        if (performanceData[i] < *minPerformance) {
            *minPerformance = performanceData[i];
        }
    }
}

int main() {
    // Example usage
    double performanceData[] = {85.5, 90.3, 78.2, 91.7, 88.9};
    int size = sizeof(performanceData) / sizeof(performanceData[0]);

    double maxPerformance, minPerformance;

    trackPerformance(performanceData, size, &maxPerformance, &minPerformance);

    printf("Max Performance: %.2f\n", maxPerformance);
    printf("Min Performance: %.2f\n", minPerformance);

    return 0;
}
```

22.Sensor Data Filter
Function Prototype: void filterSensorData(const double *sensorData, double *filteredData, int size, double filterThreshold)
Data Types: const double*, double*, int, double
Concepts: Arrays, functions, decision-making.

Details: Filter sensor data based on a threshold, updating the filtered data array by reference.

```
#include <stdio.h>

void filterSensorData(const double *sensorData, double *filteredData, int size, double filterThreshold) {
    int filteredIndex = 0;  // Index for the filteredData array

    for (int i = 0; i < size; i++) {
        if (sensorData[i] > filterThreshold) {
            filteredData[filteredIndex] = sensorData[i];
            filteredIndex++;  // Move to the next position in filteredData
        }
    }

    // Optionally, you can fill the remaining positions of filteredData with a sentinel value, if needed.
    // For example, if you want to set the remaining spots to 0 after filtering:
    for (int i = filteredIndex; i < size; i++) {
        filteredData[i] = 0.0;
    }
}

int main() {
    double sensorData[] = {1.2, 5.6, 0.9, 3.8, 4.5, 2.1};
    int size = sizeof(sensorData) / sizeof(sensorData[0]);
    double filteredData[size];  // Array to hold the filtered data

    double filterThreshold = 3.0;

    // Call the filter function
    filterSensorData(sensorData, filteredData, size, filterThreshold);

    // Output filtered data
    printf("Filtered Data:\n");
    for (int i = 0; i < size; i++) {
        if (filteredData[i] > 0.0) {  // Only print non-zero values
            printf("%.2f ", filteredData[i]);
        }
    }

    return 0;
}
```

23.Logistics Data Planner
Function Prototype: void planLogistics(const double *resourceLevels, double *logisticsPlan, int size)
Data Types: const double*, double*, int
Concepts: Arrays, functions, pointers, loops.
Details: Plan logistics based on resource levels, updating the logistics plan array by reference.

```
#include <stdio.h>

void planLogistics(const double *resourceLevels, double *logisticsPlan, int size) {
    for (int i = 0; i < size; i++) {
        // Example: If the resource level is greater than 100, assign a large logistics value
        // If the resource level is less than 50, assign a smaller logistics value
        if (resourceLevels[i] > 100.0) {
```

```
            logisticsPlan[i] = resourceLevels[i] * 0.8;  // allocate 80% of resources
        } else if (resourceLevels[i] < 50.0) {
            logisticsPlan[i] = resourceLevels[i] * 0.5;  // allocate 50% of resources
        } else {
            logisticsPlan[i] = resourceLevels[i] * 0.6;  // allocate 60% of resources
        }
    }
}

int main() {
    double resources[] = {120.0, 30.0, 75.0, 150.0};  // Example resource levels
    int size = sizeof(resources) / sizeof(resources[0]);
    double logistics[size];  // This array will store the logistics plan

    // Call the planLogistics function to calculate logistics based on resources
    planLogistics(resources, logistics, size);

    // Output the resulting logistics plan
    printf("Logistics Plan:\n");
    for (int i = 0; i < size; i++) {
        printf("Resource %d: %.2f, Logistics Plan: %.2f\n", i + 1, resources[i], logistics[i]);
    }

    return 0;
}
```

24.Satellite Image Processor
Function Prototype: void processSatelliteImage(const unsigned char *imageData, unsigned char *processedImage, int width, int height)
Data Types: const unsigned char*, unsigned char*, int
Concepts: Arrays, functions, pointers, loops.
Details: Process satellite image data, updating the processed image by reference.

```
#include <stdio.h>

// Function prototype declaration
void processSatelliteImage(const unsigned char *imageData, unsigned char *processedImage, int width, int height);

int main() {
    // Example of a simple grayscale image of 4x4 pixels (just for demonstration)
    unsigned char imageData[16] = {
        255, 128, 64, 32,
        16,  8,  4,  2,
        1,   2,  4,  8,
        16,  32, 64, 128
    };

    unsigned char processedImage[16];

    // Process the image (inverting the colors in this case)
    processSatelliteImage(imageData, processedImage, 4, 4);

    // Print the processed image (for demonstration purposes)
    printf("Processed Image:\n");
```

```
    for (int i = 0; i < 16; i++) {
        printf("%d ", processedImage[i]);
        if ((i + 1) % 4 == 0) {
            printf("\n");
        }
    }

    return 0;
}

// Function to process the satellite image
void processSatelliteImage(const unsigned char *imageData, unsigned char *processedImage, int width,
int height) {
    // Loop through each pixel in the image
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            // Calculate the index for the current pixel in the 1D array
            int index = y * width + x;

            // Example processing: Invert the grayscale color (assuming it's a grayscale image)
            processedImage[index] = 255 - imageData[index]; // Inverting the pixel value
        }
    }
}
```

25.Flight Path Analyzer
Function Prototype: void analyzeFlightPath(const double *pathCoordinates, double *optimizedPath, int size)
Data Types: const double*, double*, int
Concepts: Arrays, functions, pointers, loops.
Details: Analyze and optimize flight path coordinates, updating the optimized path by reference.

```
#include <stdio.h>

void analyzeFlightPath(const double *pathCoordinates, double *optimizedPath, int size) {
    if (size <= 1) {
        // No optimization needed if there is 0 or 1 coordinate.
        return;
    }

    // For simplicity, we average consecutive coordinates as an example of optimization.
    for (int i = 1; i < size - 1; ++i) {
        optimizedPath[i] = (pathCoordinates[i - 1] + pathCoordinates[i + 1]) / 2.0;
    }

    // Edge case: Copy the first and last coordinates directly, as they don't have two neighbors.
    optimizedPath[0] = pathCoordinates[0];
    optimizedPath[size - 1] = pathCoordinates[size - 1];
}

int main() {
    // Example flight path coordinates (could be latitude/longitude or any other coordinate system)
    double pathCoordinates[] = {100.0, 101.0, 99.5, 102.0, 104.0};
    int size = sizeof(pathCoordinates) / sizeof(pathCoordinates[0]);
```

```
    // Array to store the optimized path
    double optimizedPath[size];

    // Call the function to analyze and optimize the flight path
    analyzeFlightPath(pathCoordinates, optimizedPath, size);

    // Print the optimized path coordinates
    printf("Optimized Flight Path Coordinates:\n");
    for (int i = 0; i < size; ++i) {
        printf("%.2f ", optimizedPath[i]);
    }
    printf("\n");

    return 0;
}
```

26.AI Data Augmenter
Function Prototype: void augmentData(const double *originalData, double *augmentedData, int size, double augmentationFactor)
Data Types: const double*, double*, int, double
Concepts: Arrays, functions, pointers, loops.
Details: Augment AI data by applying an augmentation factor, updating the augmented data array by reference.

```
#include <stdio.h>

void augmentData(const double *originalData, double *augmentedData, int size, double augmentationFactor) {
    // Loop through each element in the original data array
    for (int i = 0; i < size; i++) {
        // Apply the augmentation factor to each element and store in the augmentedData array
        augmentedData[i] = originalData[i] * augmentationFactor;
    }
}

int main() {
    // Example original data
    double originalData[] = {1.0, 2.0, 3.0, 4.0, 5.0};
    int size = sizeof(originalData) / sizeof(originalData[0]);

    // Create an array to hold the augmented data
    double augmentedData[size];

    // Augmentation factor (e.g., scaling by 2)
    double augmentationFactor = 2.0;

    // Call augmentData function
    augmentData(originalData, augmentedData, size, augmentationFactor);

    // Output the augmented data
    printf("Augmented Data: \n");
    for (int i = 0; i < size; i++) {
        printf("%f ", augmentedData[i]);
    }
    printf("\n");
```

```
    return 0;
}
```

27.Medical Image Analyzer
Function Prototype: void analyzeMedicalImage(const unsigned char *imageData, unsigned char *analysisResults, int width, int height)
Data Types: const unsigned char*, unsigned char*, int
Concepts: Arrays, functions, pointers, loops.
Details: Analyze medical image data, updating analysis results by reference.

```
#include <stdio.h>

void analyzeMedicalImage(const unsigned char *imageData, unsigned char *analysisResults, int width, int height) {
    // Iterate through every pixel of the image.
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            // Calculate the index for the 1D representation of the 2D image.
            int pixelIndex = y * width + x;

            // Retrieve the pixel value (e.g., grayscale intensity).
            unsigned char pixelValue = imageData[pixelIndex];

            // Perform analysis. For simplicity, let's say we're thresholding.
            // If the pixel value is above a certain threshold (e.g., 128), classify it as "feature detected."
            if (pixelValue > 128) {
                analysisResults[pixelIndex] = 255;  // Mark as feature detected (e.g., white).
            } else {
                analysisResults[pixelIndex] = 0;    // No feature detected (e.g., black).
            }
        }
    }
}

int main() {
    // Example image data: a 5x5 image (grayscale pixel values).
    unsigned char imageData[5 * 5] = {
        255, 100, 50, 180, 30,
        200, 120, 150, 220, 60,
        90, 200, 250, 40, 70,
        180, 240, 30, 110, 160,
        70, 120, 220, 180, 130
    };

    unsigned char analysisResults[5 * 5];  // Array to store the analysis results.

    int width = 5;
    int height = 5;

    // Perform the analysis
    analyzeMedicalImage(imageData, analysisResults, width, height);

    // Print the analysis results
    printf("Analysis Results (Thresholded):\n");
```

```
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            printf("%3d ", analysisResults[y * width + x]);
        }
        printf("\n");
    }

    return 0;
}
```

28.Object Tracking System
Function Prototype: void trackObjects(const double *objectData, double *trackingResults, int size)
Data Types: const double*, double*, int
Concepts: Arrays, functions, pointers, loops.
Details: Track objects based on data, updating tracking results by reference.

```
#include <stdio.h>

void trackObjects(const double *objectData, double *trackingResults, int size) {
    for (int i = 0; i < size; i++) {
        // For example, each object moves by a fixed rate of 2 units per time step.
        // Update the tracking results array to store new positions
        trackingResults[i] = objectData[i] + 2;  // Adding 2 units of movement for simplicity
    }
}

int main() {
    // Sample object data: initial positions of 5 objects
    double objectData[] = {1.0, 2.5, 3.0, 4.5, 5.0};

    // Array to store tracking results (updated positions)
    double trackingResults[5];  // Same size as objectData

    // Track objects (update their positions)
    trackObjects(objectData, trackingResults, 5);

    // Print the updated positions
    printf("Updated Positions:\n");
    for (int i = 0; i < 5; i++) {
        printf("Object %d: %.2f\n", i + 1, trackingResults[i]);
    }

    return 0;
}
```

29.Defense Strategy Optimizer
Function Prototype: void optimizeDefenseStrategy(const double *threatLevels, double *optimizedStrategies, int size)

```
#include <stdio.h>

// Function to optimize defense strategy based on threat levels
void optimizeDefenseStrategy(const double *threatLevels, double *optimizedStrategies, int size) {
    // Scaling factor for defense strategy (you can adjust this based on your optimization logic)
    double scalingFactor = 1.5;
```

```
    // Iterate over each threat level and calculate the optimized strategy
    for (int i = 0; i < size; i++) {
        optimizedStrategies[i] = threatLevels[i] * scalingFactor; // Simple scaling
    }
}

// Main function for testing
int main() {
    // Example threat levels
    double threatLevels[] = {1.0, 2.5, 4.0, 3.3, 0.8};
    int size = sizeof(threatLevels) / sizeof(threatLevels[0]);

    // Array to store the optimized strategies
    double optimizedStrategies[size];

    // Call the optimizeDefenseStrategy function
    optimizeDefenseStrategy(threatLevels, optimizedStrategies, size);

    // Print the optimized defense strategies
    printf("Optimized Defense Strategies:\n");
    for (int i = 0; i < size; i++) {
        printf("Threat Level: %.2f -> Optimized Strategy: %.2f\n", threatLevels[i], optimizedStrategies[i]);
    }

    return 0;
}
```

NULL CHARACTER

==========

```
#include <stdio.h>

int main()
{
    char arr[]={'H','E','L','L','0'};
    for(int i=0;i<6;i++){
        printf("arr[i]=%p\n",&arr[i]);//(arr+i)
    }

    return 0;
}
```

USE OF %S

==========

```
#include <stdio.h>

int main()
{
    char arr1[15]="Hello world!";
    printf("%s",arr1);
    return 0;
}
```

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

int main()
{
    char arr1[15];
    printf("ENTER THE FIRST NAME:");
    scanf("%s",arr1);
    return 0;
}


#include <stdio.h>

int main(void)
{
    char str1[]="To be or not to be";
    char str2[]="that is the question";
    unsigned int count =0;

    while(str1[count]!='\0')
        ++count;
    printf("the length of te string \"%s"" is %d characters\n"str1,count);
    count=0;
    while(str2[count]!='\0')
        ++count;
    printf("the length of te string\"%s"" is  %d characters\n"str2,count);
    count=0;
    return 0;
}
```

QUESTIONS

==========

1.FUNCTION TO CALCULATE LENGTH OF THE STRING

```
#include <stdio.h>

int strLength(const char *str) {
    int len = 0;

    // Iterate through the string until the null character '\0' is encountered
    while (str[len] != '\0') {
        len++;  // Increment length for each character
    }

    return len;
}

int main() {
    char str[] = "Hello, world!";

    int len = strLength(str);
    printf("The length of the string is: %d\n", len);

    return 0;
```

```
}
```

2.CONCATENATION

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

// Function to concatenate two strings
void result(char *str1, const char *str2) {
    // Move the pointer to the end of the first string
    while (*str1 != '\0') {
        str1++;
    }

    // Append each character from str2 to the end of str1
    while (*str2 != '\0') {
        *str1 = *str2;
        str1++;
        str2++;
    }

    // Null-terminate the concatenated string
    *str1 = '\0';
}

int main() {
    char str1[100] = "Hello, ";  // Ensure enough space for concatenation
    char str2[] = "world!";

    // Concatenate str2 to str1
    result(str1, str2);

    printf("Concatenated String: %s\n", str1);
    return 0;
}
```

CHECKING IF TWO STRINGS ARE EQUAL OR NOT

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

int areStringsEqual(const char *str1, const char *str2) {
    // Traverse both strings and compare each character
    while (*str1 != '\0' && *str2 != '\0') {
        if (*str1 != *str2) {
            return 0; // Strings are not equal
        }
        str1++;
        str2++;
    }

    // If both strings end at the same time, they are equal
    // If one string ends before the other, they are not equal
    return *str1 == *str2;
}

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

```
    // Input two strings
    printf("Enter first string: ");
    fgets(str1, sizeof(str1), stdin);
    printf("Enter second string: ");
    fgets(str2, sizeof(str2), stdin);

    // Remove newline characters (if fgets reads them)
    for (int i = 0; str1[i] != '\0'; i++) {
        if (str1[i] == '\n') {
            str1[i] = '\0'; // Remove newline character
            break;
        }
    }

    for (int i = 0; str2[i] != '\0'; i++) {
        if (str2[i] == '\n') {
            str2[i] = '\0'; // Remove newline character
            break;
        }
    }

    // Check if strings are equal
    if (areStringsEqual(str1, str2)) {
        printf("The strings are equal.\n");
    } else {
        printf("The strings are not equal.\n");
    }

    return 0;
}
```

USAGE OF STRCPY

```
==================================
#include <stdio.h>
#include<string.h>
int main(){
    char firstname[50];
    char lastname[50];

    strcpy(firstname,"noel");
    strcpy(lastname,"suresh");
    printf("name is %s %s",firstname,lastname);
    return 0;
}
```

USE OF STENCPY

```
======================
#include <stdio.h>
#include<string.h>
int main(){
    char firstname[7];
    //char lastname[50];

    strncpy(firstname,"noel",3);
   // strcpy(lastname,"suresh");
```

```
    printf("name is %s ",firstname);
    return 0;
}
```

USE OF STRCAT

==========

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

int main(){
char firstName[20];
char lastName[10];

strncpy(firstName,"Abhinav",4);
strcpy(lastName,"karan");

printf("Name = %s \n",strcat(firstName,lastName));
printf("firstName = %s \n",firstName);
printf("lastName = %s \n",lastName);
return 0;
}
```

SET OF PROBLEMS

==========

1.String Length Calculation

Requirement: Write a program that takes a string input and calculates its length using strlen(). The program should handle empty strings and output appropriate messages.

Input: A string from the user.

Output: Length of the string.

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

int main() {
    char str[100];  // Array to store the input string

    // Prompt the user to input a string
    printf("Enter a string: ");
    scanf("%99s", stdin);  // Read a string with spaces

    // Manually remove the newline character at the end (if present)
    if (str[strlen(str) - 1] == '\n') {
        str[strlen(str) - 1] = '\0';  // Replace newline with null character
    }

    // Check if the string is empty
    if (strlen(str) == 0) {
        printf("The string is empty.\n");
    } else {
        printf("The length of the string is: %zu\n", strlen(str));
    }

    return 0;
}
```

2.String Copy
Requirement: Implement a program that copies one string to another using strcpy(). The program should validate if the source string fits into the destination buffer.
Input: Two strings from the user (source and destination).
Output: The copied string.

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

void safe_strcpy(char *dest, const char *src, size_t dest_size) {
    // Check if the source string fits in the destination buffer
    if (strlen(src) >= dest_size) {
        printf("Error: Source string is too large to fit in the destination buffer.\n");
        return;
    }
    // Perform the copy using strcpy
    strcpy(dest, src);
}

int main() {
    char source[100], destination[100];

    // Take input for the source and destination strings
    printf("Enter the source string: ");
    scanf("%99s", source); // Read up to 99 characters to avoid overflow

    printf("Enter the destination string: ");
    scanf("%99s", destination); // Read up to 99 characters to avoid overflow

    // Validate and copy the string
    safe_strcpy(destination, source, sizeof(destination));

    // Output the copied string
    printf("Copied string: %s\n", destination);

    return 0;
}
```

3.String Concatenation
Requirement: Create a program that concatenates two strings using strcat(). Ensure the destination string has enough space to hold the result.
Input: Two strings from the user.
Output: The concatenated string.

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

int main() {
    char str1[100], str2[100];  // Declare arrays to hold input strings
    char result[200];  // Array to store the concatenated result (must be large enough)

    // Take input from the user
    printf("Enter the first string: ");
    scanf("%s", str1);  // Read first string using scanf()
```

```
    printf("Enter the second string: ");
    scanf("%s", str2);  // Read second string using scanf()

    // Concatenate the two strings
    strcpy(result, str1);  // Copy the first string to the result
    strcat(result, str2);  // Concatenate the second string to the result

    // Output the concatenated result
    printf("The concatenated string is: %s\n", result);

    return 0;
}
```

4.String Comparison
Requirement: Develop a program that compares two strings using strcmp(). It should indicate if they are equal or which one is greater.
Input: Two strings from the user.
Output: Comparison result.

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

int main() {
    char str1[100], str2[100]; // Declare two character arrays to hold the input strings

    // Ask user to input two strings
    printf("Enter the first string: ");
    scanf("%s", str1);  // Read first string using scanf
    printf("Enter the second string: ");
    scanf("%s", str2);  // Read second string using scanf

    // Compare the strings using strcmp
    int result = strcmp(str1, str2);

    if (result == 0) {
        // Strings are equal
        printf("The strings are equal.\n");
    } else if (result < 0) {
        // str1 is lexicographically smaller than str2
        printf("The first string is smaller than the second string.\n");
    } else {
        // str1 is lexicographically greater than str2
        printf("The first string is greater than the second string.\n");
    }

    return 0;
}
```

5.Convert to Uppercase
Requirement: Write a program that converts all characters in a string to uppercase using strupr().
Input: A string from the user.
Output: The uppercase version of the string.

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

```
#include <string.h>

int main() {
    char str[100]; // Declare a string of size 100 (you can change the size as needed)

    // Take input from the user
    printf("Enter a string: ");
    scanf("%[^\n]%*c", str); // This allows input with spaces until Enter is pressed

    // Convert the string to uppercase using strupr
    strupr(str);

    // Display the uppercase string
    printf("Uppercase version: %s\n", str);

    return 0;
}
```

6.Convert to Lowercase
Requirement: Implement a program that converts all characters in a string to lowercase using strlwr().
Input: A string from the user.
Output: The lowercase version of the string.

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

int main() {
    char str[100];

    // Take input from the user using scanf
    printf("Enter a string: ");
    scanf("%99[^\n]", str); // This will read the entire line of text including spaces

    // Convert string to lowercase using strlwr
    strlwr(str);

    // Output the lowercase version of the string
    printf("Lowercase string: %s\n", str);

    return 0;
}
```

7.Substring Search
Requirement: Create a program that searches for a substring within a given string using strstr() and returns its starting index or an appropriate message if not found.
Input: A main string and a substring from the user.
Output: Starting index or not found message.

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

int main() {
    char mainStr[100], subStr[100];
    char *result;
```

```
    // Input the main string and the substring
    printf("Enter the main string: ");
    scanf("%[^\n]s", mainStr);  // Reads the entire line for mainStr

    // Clear the input buffer to remove the newline character left by previous input
    getchar();

    printf("Enter the substring: ");
    scanf("%[^\n]s", subStr);  // Reads the entire line for subStr

    // Use strstr to find the first occurrence of the substring
    result = strstr(mainStr, subStr);

    if (result != NULL) {
        // Calculate the index of the found substring
        int index = result - mainStr;
        printf("Substring found at index %d.\n", index);
    } else {
        // Substring not found
        printf("Substring not found.\n");
    }

    return 0;
}
```

8.Character Search
Requirement: Write a program that finds the first occurrence of a character in a string using strchr() and returns its index or indicates if not found.
Input: A string and a character from the user.
Output: Index of first occurrence or not found message.

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

int main() {
    char str[100], ch;
    char *ptr;

    // Input string and character to search
    printf("Enter a string: ");
    scanf("%s", str);  // Use scanf to input the string (without spaces)

    printf("Enter the character to find: ");
    scanf(" %c", &ch);  // Notice the space before %c to ignore any newline left by previous input

    // Use strchr to find the first occurrence of the character
    ptr = strchr(str, ch);

    if (ptr != NULL) {
        // Calculate the index by subtracting the base address of the string from the address returned by
strchr
        int index = ptr - str;
        printf("The character '%c' first occurs at index: %d\n", ch, index);
    } else {
        printf("Character '%c' not found in the string.\n", ch);
```

```
    }

    return 0;
}
```

9.String Reversal
Requirement: Implement a function that reverses a given string in place without using additional memory, leveraging strlen() for length determination.
Input: A string from the user.
Output: The reversed string.

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

void reverseString(char str[]) {
    int length = strlen(str); // Find the length of the string
    int start = 0;            // Pointer to the beginning of the string
    int end = length - 1;     // Pointer to the end of the string
    char temp;                // Temporary variable for swapping

    // Swap characters from the start and end, moving towards the center
    while (start < end) {
        // Swap the characters at the start and end
        temp = str[start];
        str[start] = str[end];
        str[end] = temp;

        // Move the pointers towards the center
        start++;
        end--;
    }
}

int main() {
    char str[100]; // Declare a string of size 100

    printf("Enter a string: ");
    scanf("%99[^\n]", str);  // Use scanf to read the string up to 99 characters

    reverseString(str);  // Call the function to reverse the string

    printf("Reversed string: %s\n", str);  // Output the reversed string

    return 0;
}
```

10.String Tokenization
Requirement: Create a program that tokenizes an input string into words using strtok() and counts how many tokens were found.
Input: A sentence from the user.
Output: Number of words (tokens).

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

```
int main() {
    char input[100];
    int tokenCount = 0;

    // Read a sentence from the user
    printf("Enter a sentence: ");
    scanf("%[^\n]%*c", input);  // This will read the entire line, including spaces, until Enter is pressed.

    // Use strtok to tokenize the string by space and punctuation (spaces, tabs, etc.)
    char *token = strtok(input, " \t\n");

    // Count tokens
    while (token != NULL) {
        tokenCount++;
        token = strtok(NULL, " \t\n");
    }

    // Print the number of tokens
    printf("Number of words (tokens): %d\n", tokenCount);

    return 0;
}
```

11.String Duplication
Requirement: Write a function that duplicates an input string (allocating new memory) using strdup() and displays both original and duplicated strings.
Input: A string from the user.
Output: Original and duplicated strings

```
#include <stdio.h>
#include <string.h>  // For strlen

int main() {
    char original[100];  // Array to hold the input string

    // Prompt the user for input
    printf("Enter a string: ");
    scanf("%99[^\n]", original);  // This reads input until newline or 99 characters

    // Manually allocate memory for the duplicated string
    int len = strlen(original);
    char duplicated[len + 1];  // +1 for null-terminator

    // Copy the original string into the duplicated string
    for (int i = 0; i < len; i++) {
        duplicated[i] = original[i];
    }
    duplicated[len] = '\0';  // Null-terminate the duplicated string

    // Print both original and duplicated strings
    printf("Original string: %s\n", original);
    printf("Duplicated string: %s\n", duplicated);

    return 0;
}
```

12.Case-Insensitive Comparison
Requirement: Develop a program to compare two strings without case sensitivity using strcasecmp() and report equality or differences.
Input: Two strings from the user.
Output: Comparison result.

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

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

    // Input strings
    printf("Enter the first string: ");
    scanf("%s", str1);

    printf("Enter the second string: ");
    scanf("%s", str2);

    // Compare the strings case-insensitively using strcasecmp
    if (strcasecmp(str1, str2) == 0) {
        printf("The strings are equal (case-insensitive comparison).\n");
    } else {
        printf("The strings are different (case-insensitive comparison).\n");
    }

    return 0;
}
```

13.String Trimming
Requirement: Implement functionality to trim leading and trailing whitespace from a given string, utilizing pointer arithmetic with strlen().
Input: A string with extra spaces from the user.
Output: Trimmed version of the string

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

void trim_string(char *str) {
    // Pointer to the first character in the string
    char *start = str;

    // Move 'start' to the first non-whitespace character (space, tab, or newline)
    while (*start == ' ' || *start == '\t' || *start == '\n') {
        start++;
    }

    // If the string is empty or consists only of spaces
    if (*start == '\0') {
        *str = '\0';  // Empty the string
        return;
    }

    // Pointer to the last character in the string
```

```
    char *end = str + strlen(str) - 1;

    // Move 'end' backward until the last non-whitespace character
    while (end > start && (*end == ' ' || *end == '\t' || *end == '\n')) {
        end--;
    }

    // Null-terminate the string after the last non-whitespace character
    *(end + 1) = '\0';

    // Shift the string to the beginning
    if (start != str) {
        while (*start) {
            *str = *start;
            str++;
            start++;
        }
        *str = '\0';
    }
}

int main() {
    char str[100];

    // Example: Reading input
    printf("Enter a string with leading and trailing spaces: ");
    scanf("%[^\n]%*c", str);  // Read a line with spaces

    printf("Original String: '%s'\n", str);

    // Trim the string
    trim_string(str);

    printf("Trimmed String: '%s'\n", str);

    return 0;
}
```

14.Find Last Occurrence of Character
Requirement: Write a program that finds the last occurrence of a character in a string using manual iteration instead of library functions, returning its index.
Input: A string and a character from the user.
Output: Index of last occurrence or not found message.

```
#include <stdio.h>

int main() {
    char str[100], ch;
    int i, lastIndex = -1;

    // Input string and character from user
    printf("Enter a string: ");
    scanf("%s", str); // Using scanf for input

    printf("Enter a character to find its last occurrence: ");
```

```
    scanf(" %c", &ch); // Notice the space before %c to consume any leading whitespace

    // Iterating through the string to find the last occurrence of the character
    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            lastIndex = i;
        }
    }

    // Checking and displaying the result
    if (lastIndex != -1) {
        printf("The last occurrence of '%c' is at index %d.\n", ch, lastIndex);
    } else {
        printf("Character '%c' not found in the string.\n", ch);
    }

    return 0;
}
```

15.Count Vowels in String
Requirement: Create a program that counts how many vowels are present in an input string by iterating through each character.
Input: A string from the user.
Output: Count of vowels

```
#include <stdio.h>

int main() {
    char str[100];
    int count = 0;
    int i = 0;

    // Prompt the user for input
    printf("Enter a string: ");

    // Read the string input using getchar() without using fgets
    while ((str[i] = getchar()) != '\n' && str[i] != EOF) {
        i++;
    }
    str[i] = '\0'; // Null terminate the string

    // Iterate through the string to count vowels
    for (int j = 0; str[j] != '\0'; j++) {
        char ch = str[j];
        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
            ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {
            count++;
        }
    }

    // Output the result
    printf("The number of vowels in the string is: %d\n", count);

    return 0;
}
```

16.Count Specific Characters
Requirement: Implement functionality to count how many times a specific character appears in an input string, allowing for case sensitivity options.
Input: A string and a character from the user.
Output: Count of occurrences.

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

int count_char_occurrences(const char *str, char ch, int case_sensitive) {
    int count = 0;
    int i = 0;

    // Traverse the string
    while (str[i] != '\0') {
        // Case-sensitive comparison
        if (case_sensitive) {
            if (str[i] == ch) {
                count++;
            }
        }
        // Case-insensitive comparison without ctype.h
        else {
            // Check if both characters are the same when case is ignored
            if ((str[i] == ch) ||
                (str[i] >= 'a' && str[i] <= 'z' && str[i] - 'a' + 'A' == ch) ||
                (str[i] >= 'A' && str[i] <= 'Z' && str[i] - 'A' + 'a' == ch)) {
                count++;
            }
        }
        i++;
    }
    return count;
}

int main() {
    char str[100], ch;
    int case_sensitive;

    // Input string and character
    printf("Enter a string: ");
    scanf("%99[^\n]", str);  // Read full line including spaces

    printf("Enter the character to count: ");
    getchar();  // To consume the newline character left by previous scanf
    scanf("%c", &ch);  // Read the character to search for

    // Ask for case sensitivity
    printf("Case sensitive? (1 for Yes, 0 for No): ");
    scanf("%d", &case_sensitive);

    // Count the occurrences
    int result = count_char_occurrences(str, ch, case_sensitive);
```

```
    // Output the result
    printf("The character '%c' appears %d times in the string.\n", ch, result);

    return 0;
}
```

17.Remove All Occurrences of Character
Requirement: Write a function that removes all occurrences of a specified character from an input string, modifying it in place.
Input: A string and a character to remove from it.
Output: Modified string without specified characters.

```
#include <stdio.h>

void remove_char(char *str, char ch) {
    int i = 0, j = 0;

    // Traverse the string
    while (str[i] != '\0') {
        if (str[i] != ch) {
            // If the current character is not the one to remove, copy it to the new position
            str[j++] = str[i];
        }
        // Move to the next character
        i++;
    }
    // Null-terminate the modified string
    str[j] = '\0';
}

int main() {
    char str[100], ch;

    // Get the input string
    printf("Enter a string: ");
    scanf("%99[^\n]", str);  // This reads the whole line, including spaces

    // Get the character to remove
    printf("Enter the character to remove: ");
    scanf(" %c", &ch);  // The space before %c is to consume any leftover newline character

    // Remove occurrences of the character
    remove_char(str, ch);

    // Print the modified string
    printf("Modified string: %s\n", str);

    return 0;
}
```

18.Check for Palindrome
Requirement: Develop an algorithm to check if an input string is a palindrome by comparing characters from both ends towards the center, ignoring case and spaces.
Input: A potential palindrome from the user.
Output: Whether it is or isn't a palindrome

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

int isPalindrome(char str[]) {
    int start = 0;
    int end = strlen(str) - 1;

    while (start < end) {
        // Skip spaces from both ends
        if (str[start] == ' ') {
            start++;
        } else if (str[end] == ' ') {
            end--;
        } else {
            // Compare characters ignoring case manually
            char startChar = str[start];
            char endChar = str[end];

            // Convert both to lowercase if needed
            if (startChar >= 'A' && startChar <= 'Z') {
                startChar = startChar + ('a' - 'A');
            }
            if (endChar >= 'A' && endChar <= 'Z') {
                endChar = endChar + ('a' - 'A');
            }

            // If they are not equal, it's not a palindrome
            if (startChar != endChar) {
                return 0;  // Not a palindrome
            }

            start++;
            end--;
        }
    }
    return 1;  // It is a palindrome
}

int main() {
    char str[100];  // Buffer to store the input string
    int i = 0;
    char ch;

    // Read input one character at a time (no fgets)
    printf("Enter a string: ");
    while ((ch = getchar()) != '\n' && ch != EOF) {
        str[i++] = ch;
    }
    str[i] = '\0';  // Null terminate the string

    if (isPalindrome(str)) {
        printf("The string is a palindrome.\n");
    } else {
        printf("The string is not a palindrome.\n");
```

```
    }

    return 0;
}
```

19.Extract Substring
Requirement: Create functionality to extract a substring based on specified start index and length parameters, ensuring valid indices are provided by users.
Input: A main string, start index, and length from the user.
Output: Extracted substring or error message for invalid indices.

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

void extractSubstring(char *mainStr, int startIndex, int length) {
    int strLength = strlen(mainStr);

    // Check for invalid start index
    if (startIndex < 0 || startIndex >= strLength) {
        printf("Error: Invalid start index.\n");
        return;
    }

    // Check if the length is valid (not negative, and doesn't exceed the string length)
    if (length < 0 || startIndex + length > strLength) {
        printf("Error: Invalid length.\n");
        return;
    }

    // Create a substring buffer
    char substring[length + 1]; // +1 for null terminator
    int i;

    // Extract the substring
    for (i = 0; i < length; i++) {
        substring[i] = mainStr[startIndex + i];
    }
    substring[i] = '\0'; // Null-terminate the substring

    // Print the extracted substring
    printf("Extracted substring: %s\n", substring);
}

int main() {
    // Input string and parameters
    char mainStr[100];
    int startIndex, length;

    // User inputs (using scanf)
    printf("Enter the main string: ");
    scanf("%s", mainStr); // Read input string
    printf("Enter the start index: ");
    scanf("%d", &startIndex); // Read start index
    printf("Enter the length of the substring: ");
    scanf("%d", &length); // Read substring length
```

```
    // Extract and print the substring
    extractSubstring(mainStr, startIndex, length);

    return 0;
}
```

20.Sort Characters in String
Requirement: Implement functionality to sort characters in an input string alphabetically, demonstrating usage of nested loops for comparison without library sorting functions.
Input: A string from the user.
Output: Sorted version of the characters in the string.

```
#include <stdio.h>

int main() {
    char str[100];
    int i, j, temp;

    // Ask user for input string
    printf("Enter a string: ");
    scanf("%s", str);  // Read input without fgets

    // Get the length of the string
    for(i = 0; str[i] != '\0'; i++);

    // Sorting the string using nested loops
    for(i = 0; str[i] != '\0'; i++) {
        for(j = i + 1; str[j] != '\0'; j++) {
            // Compare characters and swap if needed
            if(str[i] > str[j]) {
                // Swap characters
                temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }

    // Print the sorted string
    printf("Sorted string: %s\n", str);

    return 0;
}
```

21.Count Words in String
Requirement: Write code to count how many words are present in an input sentence by identifying spaces as delimiters, utilizing strtok().
Input: A sentence from the user.
- Output: Number of words counted.

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

int main() {
```

```
    char sentence[1000];  // Buffer for user input
    char *token;          // Pointer for each word/token

    // Prompt user for input
    printf("Enter a sentence: ");
    scanf("%[^\n]", sentence);  // Reads until newline is encountered

    int word_count = 0;

    // Tokenize the sentence by spaces (default delimiter is space)
    token = strtok(sentence, " ");

    while (token != NULL) {
        word_count++;
        token = strtok(NULL, " ");  // Get next token
    }

    // Output the number of words
    printf("Number of words: %d\n", word_count);

    return 0;
}
```

22.Remove Duplicates from String

- Requirement: Develop an algorithm to remove duplicate characters while maintaining their first occurrence order in an input string.
- Input: A string with potential duplicate characters.
- Output: Modified version of the original without duplicates.

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

void removeDuplicates(char str[]) {
    int n = strlen(str);
    if (n == 0) return;

    // Boolean array to track the occurrence of characters
    int seen[256] = {0};  // Assuming extended ASCII characters

    int j = 0;  // j is used to track the position in the modified string
    for (int i = 0; i < n; i++) {
        // If character is not seen before, add it to the result string
        if (seen[(int)str[i]] == 0) {
            str[j++] = str[i];
            seen[(int)str[i]] = 1;
        }
    }

    // Null-terminate the string after the last unique character
    str[j] = '\0';
}

int main() {
    // Example usage
    char str[] = "programming";
```

```
    printf("Original string: %s\n", str);
    removeDuplicates(str);
    printf("String after removing duplicates: %s\n", str);
    return 0;
}
```

23.Find First Non-Repeating Character
- Requirement: Create functionality to find the first non-repeating character in an input string, demonstrating effective use of arrays for counting occurrences.
- Input: A sample input from the user.
- Output: The first non-repeating character or indication if all are repeating.

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

#define MAX_CHAR 256

// Function to find the first non-repeating character
char firstNonRepeatingCharacter(const char* str) {
    int count[MAX_CHAR] = {0}; // Array to store character counts
    int i;

    // Count the occurrences of each character
    for (i = 0; str[i] != '\0'; i++) {
        count[(unsigned char)str[i]]++;
    }

    // Find the first character that has a count of 1
    for (i = 0; str[i] != '\0'; i++) {
        if (count[(unsigned char)str[i]] == 1) {
            return str[i]; // Return the first non-repeating character
        }
    }

    return '\0'; // Return null character if no non-repeating character is found
}

int main() {
    char str[100];

    // Get user input
    printf("Enter a string: ");
    scanf("%s", str); // Using scanf to get input instead of fgets

    char result = firstNonRepeatingCharacter(str);

    if (result == '\0') {
        printf("All characters are repeating.\n");
    } else {
        printf("The first non-repeating character is: %c\n", result);
    }

    return 0;
}
```

24.Convert String to Integer
- Requirement: Implement functionality to convert numeric strings into integer values without using standard conversion functions like atoi(), handling invalid inputs gracefully.
- Input: A numeric string.
- Output: Converted integer value or error message.

```
#include <stdio.h>

int stringToInt(const char *str) {
    int result = 0;  // This will store the converted integer
    int sign = 1;    // To handle negative numbers, if any

    // Check for empty string
    if (str == NULL || *str == '\0') {
        printf("Error: Invalid input (empty string).\n");
        return -1;
    }

    // Handle optional leading '+' or '-' sign
    if (*str == '-') {
        sign = -1;
        str++;  // Move to the next character
    } else if (*str == '+') {
        str++;  // Skip the '+' sign
    }

    // Iterate through each character
    while (*str != '\0') {
        // Check if the character is a valid digit by comparing ASCII values
        if (*str < '0' || *str > '9') {
            printf("Error: Invalid input (non-numeric character encountered).\n");
            return -1;
        }

        // Convert the current character to a digit and add to the result
        result = result * 10 + (*str - '0');
        str++;  // Move to the next character
    }

    // Apply the sign
    return result * sign;
}

int main() {
    char input[100];

    // Ask the user for input
    printf("Enter a numeric string: ");
    if (scanf("%s", input) != 1) {
        printf("Error: Invalid input.\n");
        return -1;
    }

    // Convert string to integer
    int value = stringToInt(input);
```

```
    // Output the result
    if (value != -1) {
        printf("Converted integer: %d\n", value);
    }

    return 0;
}
```

25.Check Anagram Status Between Two Strings
- Requirement: Write code to check if two strings are anagrams by sorting their characters and comparing them.
- Input: Two strings.
- Output: Whether they are anagrams.

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

// Function to sort a string
void sortString(char str[]) {
    int n = strlen(str);
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (str[i] > str[j]) {
                // Swap characters
                char temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }
}

// Function to check if two strings are anagrams
int areAnagrams(char str1[], char str2[]) {
    // If lengths are different, they cannot be anagrams
    if (strlen(str1) != strlen(str2)) {
        return 0;
    }

    // Sort both strings
    sortString(str1);
    sortString(str2);

    // Compare sorted strings
    for (int i = 0; i < strlen(str1); i++) {
        if (str1[i] != str2[i]) {
            return 0;  // Not an anagram
        }
    }
    return 1;  // Strings are anagrams
}

int main() {
```

```
    char str1[100], str2[100];

    // Take input for the two strings
    printf("Enter the first string: ");
    scanf("%s", str1);  // Read a single word
    printf("Enter the second string: ");
    scanf("%s", str2);  // Read a single word

    // Convert both strings to lowercase to ignore case
    for (int i = 0; str1[i]; i++) {
        str1[i] = tolower(str1[i]);
    }
    for (int i = 0; str2[i]; i++) {
        str2[i] = tolower(str2[i]);
    }

    // Check if the strings are anagrams
    if (areAnagrams(str1, str2)) {
        printf("The strings are anagrams.\n");
    } else {
        printf("The strings are not anagrams.\n");
    }

    return 0;
}
```

26.Merge Two Strings Alternately
- Requirement: Create functionality to merge two strings alternately into one while handling cases where strings may be of different lengths.
- Input: Two strings.
- Output: Merged alternating characters.

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

void mergeStringsAlternately(char *str1, char *str2, char *result) {
    int i = 0, j = 0, k = 0;
    int len1 = strlen(str1), len2 = strlen(str2);

    // Merge the strings alternately
    while (i < len1 && j < len2) {
        result[k++] = str1[i++];
        result[k++] = str2[j++];
    }

    // Append remaining characters from the longer string
    while (i < len1) {
        result[k++] = str1[i++];
    }

    while (j < len2) {
        result[k++] = str2[j++];
    }

    // Null-terminate the result string
```

```
    result[k] = '\0';
}

int main() {
    char str1[100], str2[100], result[200];

    // Taking input for the two strings
    printf("Enter first string: ");
    scanf("%s", str1);  // No fgets here, using scanf instead
    printf("Enter second string: ");
    scanf("%s", str2);  // No fgets here, using scanf instead

    // Merging the strings alternately
    mergeStringsAlternately(str1, str2, result);

    // Output the merged string
    printf("Merged string: %s\n", result);

    return 0;
}
```

27.Count Consonants in String
- Requirement: Develop code to count consonants while ignoring vowels and whitespace characters.
- Input: Any input text.
- Output: Count of consonants.

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

int main() {
    char ch;
    int consonantCount = 0;

    printf("Enter a string (Ctrl+D to end input):\n");

    // Reading input character by character
    while ((ch = getchar()) != EOF) {
        // Convert character to lowercase to simplify vowel checking
        ch = tolower(ch);

        // Check if the character is a letter and is not a vowel
        if ((ch >= 'a' && ch <= 'z') && !(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')) {
            consonantCount++;
        }
    }

    // Output the result
    printf("Consonant count: %d\n", consonantCount);

    return 0;
}
```

28.Replace Substring with Another String
- Requirement: Write functionality to replace all occurrences of one substring with another within a given main string.

- Input: Main text, target substring, replacement substring.
- Output: Modified main text after replacements.

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

// Function to replace all occurrences of target substring with replacement substring
void replaceSubstring(char *mainText, const char *target, const char *replacement) {
    // Find the size of the main text, target, and replacement
    int mainLen = strlen(mainText);
    int targetLen = strlen(target);
    int replacementLen = strlen(replacement);

    // If target is empty, do nothing
    if (targetLen == 0) {
        return;
    }

    // Allocate enough memory for the worst-case scenario
    char *result = malloc(mainLen + 1);
    int i = 0, j = 0;

    while (i < mainLen) {
        // Check if we found the target substring
        if (strncmp(&mainText[i], target, targetLen) == 0) {
            // Replace with the replacement string
            strcpy(&result[j], replacement);
            i += targetLen; // Move past the target substring
            j += replacementLen; // Move past the replacement string
        } else {
            // Otherwise, just copy the current character
            result[j++] = mainText[i++];
        }
    }

    result[j] = '\0'; // Null-terminate the result

    // Copy the result back to the main text
    strcpy(mainText, result);

    // Free the allocated memory for the result
    free(result);
}

int main() {
    char mainText[1024] = "This is the original text with a word to replace.";
    const char *target = "replace";
    const char *replacement = "substitute";

    printf("Original text: %s\n", mainText);

    // Call the function to replace substring
    replaceSubstring(mainText, target, replacement);
```

```
    printf("Modified text: %s\n", mainText);

    return 0;
}
```

29.Count Occurrences of Substring
- Requirement: Create code that counts how many times one substring appears within another larger main text without overlapping occurrences.
- Input: Main text and target substring.
- Output: Count of occurrences

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

int count_occurrences(const char *main_text, const char *target_substring) {
    int count = 0;
    const char *temp = main_text;

    while ((temp = strstr(temp, target_substring)) != NULL) {
        count++;
        temp += strlen(target_substring); // Move past the last found substring
    }

    return count;
}

int main() {
    const char *main_text = "This is a test text. Test it well. Testing is important.";
    const char *target_substring = "Test";

    int occurrences = count_occurrences(main_text, target_substring);
    printf("The substring \"%s\" appears %d times.\n", target_substring, occurrences);

    return 0;
}
```

30.mplement Custom String Length Function
- Requirement: Finally, write your own implementation of strlen() function from scratch, demonstrating pointer manipulation techniques.
- Input: Any input text.
- Output: Length calculated by custom function.

```
#include <stdio.h>

// Custom implementation of strlen using pointers
size_t custom_strlen(const char *str) {
    const char *ptr = str;  // Pointer to the start of the string
    size_t length = 0;

    // Iterate through the string until we find the null terminator
    while (*ptr != '\0') {
        length++;  // Increment length for each character
        ptr++;     // Move the pointer to the next character
    }
```

```
    return length;
}

int main() {
    // Test the custom_strlen function
    const char *input_string = "Hello, world!";
    size_t length = custom_strlen(input_string);

    printf("Length of the string: %zu\n", length);

    return 0;
}
```