Day 10 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 <math.h>


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

void computeStats(const double *array, int size, double *average, double *variance) {
    double sum = 0, sumSq = 0;
    for (int i = 0; i < size; i++) {
        sum += array[i];
        sumSq += array[i] * array[i];
    }
    *average = sum / size;
    *variance = (sumSq / size) - (*average) * (*average);
}

int main() {
    double data[] = {2.5, 3.5, 5.0, 4.5};
    int size = 4;
    double average, variance;

    computeStats(data, size, &average, &variance);
```

```
    printf("Average: %.2lf, Variance: %.2lf\n", average, variance);

    return 0;
}
```

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>

double* normalizeData(const double *array, int size) {
    double *normalized = (double*)malloc(size * sizeof(double));
    double max = array[0], min = array[0];
    for (int i = 1; i < size; i++) {
        if (array[i] > max) max = array[i];
        if (array[i] < min) min = array[i];
    }
    for (int i = 0; i < size; i++) {
        normalized[i] = (array[i] - min) / (max - min);
    }
    return normalized;
}

int main() {
```

```
    double data[] = {2.5, 3.5, 5.0, 4.5};
    int size = 4;

    double *normalized = normalizeData(data, size);
    printf("Normalized Data:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", normalized[i]);
    }
    printf("\n");

    free(normalized);
    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) {
    for (int i = 0; i < size; i++) {
        printf("Result: %.2lf, Description: %s\n", results[i], descriptions[i]);
    }
}
```

```
int main() {
    double results[] = {2.5, 3.5, 4.5};
    const char *descriptions[] = {"Test 1", "Test 2", "Test 3"};
    int size = 3;

    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) {
    *anomalyCount = 0;
    for (int i = 0; i < size; i++) {
        if (data[i] > threshold) {
            (*anomalyCount)++;
        }
    }
}
```

```
int main() {
    double data[] = {1.5, 2.5, 3.5, 5.0};
    int size = 4;
    double threshold = 3.0;
    int anomalyCount;

    detectAnomalies(data, size, threshold, &anomalyCount);
    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>
#include <string.h>

void classifyData(const double *data, int size, char *labels[], double threshold) {
    for (int i = 0; i < size; i++) {
        if (data[i] >= threshold) {
            labels[i] = "High";
        } else {
            labels[i] = "Low";
        }
```

```
    }
}

int main() {
    double data[] = {1.5, 3.5, 5.5, 2.0};
    int size = 4;
    double threshold = 3.0;
    char *labels[size];

    classifyData(data, size, labels, threshold);
    printf("Data Classification:\n");
    for (int i = 0; i < size; i++) {
        printf("Data: %.2lf, Label: %s\n", data[i], labels[i]);
    }

    return 0;
}
```

6.Artificial Intelligence

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>

void adjustWeights(double *weights, int size, double learningRate) {
```

```
    for (int i = 0; i < size; i++) {
        weights[i] += learningRate * weights[i];
    }
}

int main() {
    double weights[] = {0.5, 1.0, 1.5};
    int size = 3;
    double learningRate = 0.1;

    adjustWeights(weights, size, learningRate);
    printf("Adjusted Weights:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", weights[i]);
    }
    printf("\n");

    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) {
    *bestAccuracy = accuracies[0];
    for (int i = 1; i < size; i++) {
        if (accuracies[i] > *bestAccuracy) {
            *bestAccuracy = accuracies[i];
        }
    }
}

int main() {
    double accuracies[] = {85.5, 90.0, 88.0, 92.5};
    int size = 4;
    double bestAccuracy;

    evaluateModels(accuracies, size, &bestAccuracy);
    printf("Best Accuracy: %.2lf%%\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>
```

```
void constructDecisionTree(const double *features, int size, int *treeStructure) {
    for (int i = 0; i < size; i++) {
        treeStructure[i] = (features[i] > 0.5) ? 1 : 0; // Simple binary split example
    }
}

int main() {
    double features[] = {0.3, 0.7, 0.4, 0.8};
    int size = 4;
    int treeStructure[size];

    constructDecisionTree(features, size, treeStructure);
    printf("Decision Tree Structure:\n");
    for (int i = 0; i < size; i++) {
        printf("%d ", treeStructure[i]);
    }
    printf("\n");

    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.

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

```
#include <string.h>

void processSentiments(const char *sentences[], int size, int *sentimentScores) {
    for (int i = 0; i < size; i++) {
        // Example: Positive sentiment if "good" is found
        sentimentScores[i] = (strstr(sentences[i], "good") != NULL) ? 1 : -1;
    }
}

int main() {
    const char *sentences[] = {"This is a good day.", "I feel bad about this.", "Good things are happening."};
    int size = 3;
    int sentimentScores[size];

    processSentiments(sentences, size, sentimentScores);
    printf("Sentiment Scores:\n");
    for (int i = 0; i < size; i++) {
        printf("Sentence: %s, Score: %d\n", sentences[i], 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>
#include <stdlib.h>

double* generateTrainingData(const double *baseData, int size, int multiplier) {
    double *trainingData = (double*)malloc(size * multiplier * sizeof(double));
    for (int i = 0; i < size * multiplier; i++) {
        trainingData[i] = baseData[i % size] * (i / size + 1);
    }
    return trainingData;
}

int main() {
    double baseData[] = {1.0, 2.0, 3.0};
    int size = 3, multiplier = 2;

    double *trainingData = generateTrainingData(baseData, size, multiplier);
    printf("Generated Training Data:\n");
    for (int i = 0; i < size * multiplier; i++) {
        printf("%.2lf ", trainingData[i]);
    }
    printf("\n");

    free(trainingData);
    return 0;
}
```

11.Computer Vision

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) {
    for (int i = 0; i < width * height; i++) {
        filteredImage[i] = 255 - image[i]; // Simple inversion filter
    }
}

int main() {
    unsigned char image[] = {100, 150, 200, 50, 75, 125};
    int width = 2, height = 3;
    unsigned char filteredImage[6];

    applyFilter(image, filteredImage, width, height);
    printf("Filtered Image:\n");
    for (int i = 0; i < width * height; i++) {
        printf("%d ", filteredImage[i]);
    }
    printf("\n");

    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 <stdio.h>

void detectEdges(const unsigned char *image, unsigned char *edges, int width, int height) {
    for (int i = 0; i < width * height; i++) {
        edges[i] = (image[i] > 128) ? 255 : 0; // Simple threshold-based edge detection
    }
}

int main() {
    unsigned char image[] = {100, 150, 200, 50, 75, 125};
    int width = 2, height = 3;
    unsigned char edges[6];

    detectEdges(image, edges, width, height);
    printf("Edges Detected:\n");
    for (int i = 0; i < width * height; i++) {
        printf("%d ", edges[i]);
    }
    printf("\n");

    return 0;
```

```
}
```

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>

void recognizeObjects(const double *features, int size, char *objectLabels[]) {
    for (int i = 0; i < size; i++) {
        objectLabels[i] = (features[i] > 0.5) ? "Object A" : "Object B";
    }
}

int main() {
    double features[] = {0.3, 0.7, 0.4, 0.8};
    int size = 4;
    char *objectLabels[size];

    recognizeObjects(features, size, objectLabels);
    printf("Recognized Objects:\n");
    for (int i = 0; i < size; i++) {
        printf("Feature: %.2lf, Label: %s\n", features[i], objectLabels[i]);
    }
```

```
    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>

void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth, int originalHeight, int newWidth, int newHeight) {
    for (int i = 0; i < newHeight; i++) {
        for (int j = 0; j < newWidth; j++) {
            int srcX = j * originalWidth / newWidth;
            int srcY = i * originalHeight / newHeight;
            outputImage[i * newWidth + j] = inputImage[srcY * originalWidth + srcX];
        }
    }
}

int main() {
    unsigned char inputImage[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int originalWidth = 3, originalHeight = 3;
    int newWidth = 2, newHeight = 2;
    unsigned char outputImage[4];
```

```
    resizeImage(inputImage, outputImage, originalWidth, originalHeight, newWidth,
newHeight);
    printf("Resized Image:\n");
    for (int i = 0; i < newWidth * newHeight; i++) {
        printf("%d ", outputImage[i]);
    }
    printf("\n");

    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>

void balanceColors(const unsigned char *image, unsigned char *balancedImage, int
width, int height) {
    for (int i = 0; i < width * height; i++) {
        balancedImage[i] = (image[i] + 50 > 255) ? 255 : image[i] + 50; // Example:
Increase brightness
    }
}

int main() {
```

```
    unsigned char image[] = {100, 150, 200, 50, 75, 125};
    int width = 2, height = 3;
    unsigned char balancedImage[6];

    balanceColors(image, balancedImage, width, height);
    printf("Balanced Colors:\n");
    for (int i = 0; i < width * height; i++) {
        printf("%d ", balancedImage[i]);
    }
    printf("\n");

    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.

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

void recognizePatterns(const char *patterns[], int size, int *matchCounts) {
    for (int i = 0; i < size; i++) {
        matchCounts[i] = (strstr(patterns[i], "match") != NULL) ? 1 : 0;
    }
}
```

```
int main() {
    const char *patterns[] = {"match this pattern", "no match here", "perfect match"};
    int size = 3;
    int matchCounts[size];

    recognizePatterns(patterns, size, matchCounts);
    printf("Pattern Matches:\n");
    for (int i = 0; i < size; i++) {
        printf("Pattern: %s, Match: %d\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.

```
#include <stdio.h>

void analyzeClimateData(const double *temperatureReadings, int size, double *minTemp, double *maxTemp) {
    *minTemp = *maxTemp = temperatureReadings[0];
    for (int i = 1; i < size; i++) {
        if (temperatureReadings[i] < *minTemp) *minTemp = temperatureReadings[i];
```

```
        if (temperatureReadings[i] > *maxTemp) *maxTemp = temperatureReadings[i];
    }
}

int main() {
    double temperatureReadings[] = {30.5, 25.3, 27.8, 33.2, 28.0};
    int size = 5;
    double minTemp, maxTemp;

    analyzeClimateData(temperatureReadings, size, &minTemp, &maxTemp);
    printf("Minimum Temperature: %.2lf\n", minTemp);
    printf("Maximum Temperature: %.2lf\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>

void processQuantumData(const double *measurements, int size, double *processedData) {
    for (int i = 0; i < size; i++) {
        processedData[i] = measurements[i] * 0.5; // Example: Scale down the data
```

```
    }
}

int main() {
    double measurements[] = {10.0, 20.0, 30.0, 40.0};
    int size = 4;
    double processedData[size];

    processQuantumData(measurements, size, processedData);
    printf("Processed Data:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", processedData[i]);
    }
    printf("\n");

    return 0;
}
```

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) {
    printf("Visualization: %s\n", title);
    for (int i = 0; i < size; i++) {
```

```
        printf("%.2lf ", data[i]);
    }
    printf("\n");
}

int main() {
    double data[] = {1.0, 2.0, 3.5, 4.0};
    int size = 4;
    const char *title = "Sample 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>

double* simulateGeneticData(const double *initialData, int size, double mutationRate) {
    double *simulatedData = (double*)malloc(size * sizeof(double));
    for (int i = 0; i < size; i++) {
        simulatedData[i] = initialData[i] + mutationRate * i; // Example mutation logic
```

```
    }
    return simulatedData;
}

int main() {
    double initialData[] = {1.0, 2.0, 3.0};
    int size = 3;
    double mutationRate = 0.2;

    double *simulatedData = simulateGeneticData(initialData, size, mutationRate);
    printf("Simulated Genetic Data:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", simulatedData[i]);
    }
    printf("\n");

    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) {
    *maxPerformance = *minPerformance = performanceData[0];
    for (int i = 1; i < size; i++) {
        if (performanceData[i] > *maxPerformance) *maxPerformance =
performanceData[i];
        if (performanceData[i] < *minPerformance) *minPerformance =
performanceData[i];
    }
}

int main() {
    double performanceData[] = {85.5, 90.2, 78.9, 88.1};
    int size = 4;
    double maxPerformance, minPerformance;

    trackPerformance(performanceData, size, &maxPerformance, &minPerformance);
    printf("Maximum Performance: %.2lf\n", maxPerformance);
    printf("Minimum Performance: %.2lf\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) {
    for (int i = 0; i < size; i++) {
        filteredData[i] = (sensorData[i] > filterThreshold) ? sensorData[i] : 0;
    }
}


int main() {
    double sensorData[] = {10.5, 20.3, 5.8, 15.2, 30.1};
    int size = 5;
    double filterThreshold = 15.0;
    double filteredData[size];


    filterSensorData(sensorData, filteredData, size, filterThreshold);
    printf("Filtered Data:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", filteredData[i]);
    }
    printf("\n");


    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++) {
        logisticsPlan[i] = resourceLevels[i] * 1.1; // Example: Increase by 10%
    }
}

int main() {
    double resourceLevels[] = {100.0, 200.0, 150.0};
    int size = 3;
    double logisticsPlan[size];

    planLogistics(resourceLevels, logisticsPlan, size);
    printf("Logistics Plan:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", logisticsPlan[i]);
    }
    printf("\n");

    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>

void processSatelliteImage(const unsigned char *imageData, unsigned char *processedImage, int width, int height) {
    for (int i = 0; i < width * height; i++) {
        processedImage[i] = imageData[i] / 2; // Example: Halve the intensity
    }
}

int main() {
    unsigned char imageData[] = {100, 200, 150, 250};
    int width = 2, height = 2;
    unsigned char processedImage[4];

    processSatelliteImage(imageData, processedImage, width, height);
    printf("Processed Image:\n");
    for (int i = 0; i < width * height; i++) {
        printf("%d ", processedImage[i]);
    }
    printf("\n");

    return 0;
}
```

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) {
    for (int i = 0; i < size; i++) {
        optimizedPath[i] = pathCoordinates[i] * 0.9; // Example: Reduce each coordinate by 10%
    }
}

int main() {
    double pathCoordinates[] = {100.0, 200.0, 150.0, 250.0};
    int size = 4;
    double optimizedPath[size];

    analyzeFlightPath(pathCoordinates, optimizedPath, size);
    printf("Optimized Flight Path:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", 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) {
    for (int i = 0; i < size; i++) {
        augmentedData[i] = originalData[i] * augmentationFactor;
    }
}

int main() {
    double originalData[] = {10.0, 20.0, 30.0};
    int size = 3;
    double augmentationFactor = 1.5;
    double augmentedData[size];

    augmentData(originalData, augmentedData, size, augmentationFactor);
    printf("Augmented Data:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", 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) {
    for (int i = 0; i < width * height; i++) {
        analysisResults[i] = (imageData[i] > 128) ? 255 : 0; // Example: Threshold analysis
    }
}

int main() {
    unsigned char imageData[] = {100, 150, 200, 50};
    int width = 2, height = 2;
    unsigned char analysisResults[4];

    analyzeMedicalImage(imageData, analysisResults, width, height);
    printf("Analysis Results:\n");
```

```
    for (int i = 0; i < width * height; i++) {
        printf("%d ", analysisResults[i]);
    }
    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++) {
        trackingResults[i] = objectData[i] * 1.2; // Example: Increase values by 20%
    }
}

int main() {
    double objectData[] = {50.0, 75.0, 100.0};
    int size = 3;
    double trackingResults[size];

    trackObjects(objectData, trackingResults, size);
```

```
    printf("Tracking Results:\n");
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", trackingResults[i]);
    }
    printf("\n");

    return 0;
}
```

29.Defense Strategy Optimizer

Function Prototype: void optimizeDefenseStrategy(const double *threatLevels, double *optimizedStrategies, int size)

```
#include <stdio.h>

void optimizeDefenseStrategy(const double *threatLevels, double *optimizedStrategies, int size) {
    for (int i = 0; i < size; i++) {
        optimizedStrategies[i] = threatLevels[i] / 2; // Example: Reduce threat level by half
    }
}

int main() {
    double threatLevels[] = {100.0, 200.0, 150.0};
    int size = 3;
    double optimizedStrategies[size];

    optimizeDefenseStrategy(threatLevels, optimizedStrategies, size);
    printf("Optimized Strategies:\n");
```

```
    for (int i = 0; i < size; i++) {
        printf("%.2lf ", optimizedStrategies[i]);
    }
    printf("\n");

    return 0;
}
```

String 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 input[40];
    printf("Enter the String:");
    scanf("%s",input);
    int length=strlen(input);
    if(length==0){
        printf("Empty String");
    }
    else{
```

```
        printf("The length of the string is %d\n",length);
    }
    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>


int main() {
    char src[10];
    char dest[10];
    printf("Enter the Source:");
    scanf("%s",src);
    strcpy(dest,src);
    printf("The copied string is %s\n",dest);
    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 input1[20];
    char input2[20];
    printf("Enter the input1:");
    scanf("%s",input1);
    printf("Enter the input2:");
    scanf("%s",input2);
    printf("The concatinated string is %s\n",strcat(input1, input2));
    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 input1[20];
    char input2[20];
    printf("Enter the input1:");
    scanf("%s",input1);
    printf("Enter the input2:");
```

```
    scanf("%s",input2);
    int result=strcmp(input1,input2);
    if(result==0){
        printf("Both are equal");
    }
    else if(result>0){
        printf("First string is greater");
    }
    else{
        printf("Second string is greater");
    }
    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];


    printf("Enter a string: ");
    scanf("%[^\n]%*c", str);
```

```
    strupr(str);
    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];

    printf("Enter a string: ");
    scanf("%[^\n]%*c", str);

    strlwr(str);
    printf("Uppercase version: %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[50];



    printf("Enter the main string: ");
    scanf("%s", mainStr);
    printf("Enter the sub string: ");
    scanf("%s", subStr);

    char *result = strstr(mainStr, subStr);

    // Check if substring is found
    if (result != NULL) {
        printf("Substring found at index: %ld\n", result - mainStr);
    } else {
        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];
    char ch;

    // Input the string
    printf("Enter the string: ");
    scanf("%s",str);

    // Input the character to search for
    printf("Enter the character to search for: ");
    scanf("%c", &ch);

    // Find the first occurrence of the character using strchr
    char *result = strchr(str, ch);

    if (result != NULL) {
        // Calculate the index of the first occurrence
        int index = result - str;
        printf("First occurrence of '%c' is at index: %d\n", ch, index);
    } else {
        printf("Character not found.\n");
    }
```

```
    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 start = 0;
    int end = strlen(str) - 1;

    // Swap characters from start to end
    while (start < end) {
        // Swap characters
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;

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

```
int main() {
    char str[100];

    // Input the string
    printf("Enter the string: ");
    scanf("%s",str);

    // Reverse the string in place
    reverseString(str);

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

    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 str[100];
    int count = 0;

    // Input the sentence
```

```
    printf("Enter a sentence: ");
    scanf("%s", str);  // Read the entire line including spaces

    // Tokenize the string using space as a delimiter
    char *token = strtok(str, " ");

    // Count the tokens
    while (token != NULL) {
        count++;  // Increase the token count
        token = strtok(NULL, " ");  // Get the next token
    }

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

    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>

int main() {
    char str[100];
```

```
    printf("Enter a string: ");
    scanf("%[^\n]s", str);  // Read string with spaces

    // Duplicate the string
    char *dupStr = strdup(str);

    // Output the original and duplicated strings
    printf("Original string: %s\n", str);
    printf("Duplicated string: %s\n", dupStr);

    // Free allocated memory
    free(dupStr);

    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];

    printf("Enter first string: ");
```

```
    scanf("%[^\n]s", str1);
    getchar();  // Consume the newline character left by scanf
    printf("Enter second string: ");
    scanf("%[^\n]s", str2);

    // Compare the strings case-insensitively
    int result = strcasecmp(str1, str2);

    if(result == 0) {
        printf("Strings are equal (case-insensitive)\n");
    } else {
        printf("Strings are different (case-insensitive)\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>
#include <ctype.h>

void trim(char *str) {
    int start = 0, end = strlen(str) - 1;
```

```
    // Trim leading spaces
    while (isspace(str[start])) {
        start++;
    }

    // Trim trailing spaces
    while (end >= start && isspace(str[end])) {
        end--;
    }

    // Create a new string with the trimmed version
    for (int i = 0; i <= end - start; i++) {
        str[i] = str[start + i];
    }
    str[end - start + 1] = '\0';
}

int main() {
    char str[100];

    printf("Enter a string with extra spaces: ");
    scanf("%[^\n]s", str);

    trim(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>
#include <string.h>

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

    printf("Enter a string: ");
    scanf("%[^\n]s", str);
    getchar();  // Consume the newline character
    printf("Enter a character to search: ");
    scanf("%c", &ch);

    int lastIndex = -1;
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            lastIndex = i;
        }
    }

    if (lastIndex != -1) {
```

```
        printf("Last occurrence of '%c' is at index %d\n", ch, lastIndex);
    } else {
        printf("Character not found\n");
    }

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

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

    printf("Enter a string: ");
    scanf("%[^\n]s", str);

    for (int i = 0; str[i] != '\0'; i++) {
        char ch = tolower(str[i]);
        if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            vowelCount++;
```

```
        }
    }

    printf("Number of vowels: %d\n", vowelCount);

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

    printf("Enter a string: ");
    scanf("%[^\n]s", str);
    getchar();  // Consume newline
    printf("Enter the character to count: ");
    scanf("%c", &ch);

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

```
        if (str[i] == ch) {
            count++;
        }
    }

    printf("Character '%c' appears %d times\n", ch, count);

    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>
#include <string.h>

void removeChar(char *str, char ch) {
    int i = 0, j = 0;
    while (str[i]) {
        if (str[i] != ch) {
            str[j++] = str[i];
        }
        i++;
    }
    str[j] = '\0';
}
```

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

    printf("Enter a string: ");
    scanf("%[^\n]s", str);
    getchar();  // Consume newline
    printf("Enter the character to remove: ");
    scanf("%c", &ch);

    removeChar(str, ch);

    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>
#include <ctype.h>

int isPalindrome(char str[]) {
```

```
    int start = 0, end = strlen(str) - 1;

    while (start < end) {
        if (tolower(str[start]) != tolower(str[end])) {
            return 0;
        }
        start++;
        end--;
    }
    return 1;
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%[^\n]s", str);

    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 str[], int start, int length) {
    char subStr[100];
    int i;

    for (i = 0; i < length && str[start + i] != '\0'; i++) {
        subStr[i] = str[start + i];
    }
    subStr[i] = '\0';

    printf("Extracted substring: %s\n", subStr);
}

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

    printf("Enter a string: ");
    scanf("%[^\n]s", str);
    printf("Enter start index and length: ");
    scanf("%d %d", &start, &length);

    extractSubstring(str, start, 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>
#include <string.h>

void sortString(char str[]) {
    int n = strlen(str);
    char temp;

    // Simple bubble sort to sort characters in string
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (str[i] > str[j]) {
                temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }
}
```

```
int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%[^\n]s", str);

    sortString(str);

    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>
#include <ctype.h>

int countWords(char str[]) {
    int count = 0;
    int inWord = 0;

    for (int i = 0; str[i] != '\0'; i++) {
        if (isspace(str[i])) {
```

```
            inWord = 0;
        } else if (!inWord) {
            inWord = 1;
            count++;
        }
    }

    return count;
}

int main() {
    char str[100];

    printf("Enter a sentence: ");
    scanf("%[^\n]s", str);

    int wordCount = countWords(str);

    printf("Number of words: %d\n", wordCount);

    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.

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

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

    for (int i = 0; i < n; i++) {
        int j;
        for (j = 0; j < i; j++) {
            if (str[i] == str[j]) {
                break;
            }
        }
        if (j == i) {
            str[index++] = str[i];
        }
    }
    str[index] = '\0';
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%[^\n]s", str);

    removeDuplicates(str);
```

```
    printf("String without 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>

char findFirstNonRepeating(char str[]) {
    int n = strlen(str);

    for (int i = 0; i < n; i++) {
        int j;
        for (j = 0; j < n; j++) {
            if (i != j && str[i] == str[j]) {
                break;
            }
        }
        if (j == n) {
            return str[i];
        }
    }
```

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

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%[^\n]s", str);

    char result = findFirstNonRepeating(str);

    if (result != '\0') {
        printf("First non-repeating character: %c\n", result);
    } else {
        printf("No non-repeating character found\n");
    }

    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>
#include <ctype.h>
```

```
int stringToInt(char str[]) {
    int result = 0, i = 0;

    // Convert the string to integer
    while (str[i] != '\0') {
        if (!isdigit(str[i])) {
            return -1; // Invalid input (non-digit character)
        }
        result = result * 10 + (str[i] - '0');
        i++;
    }

    return result;
}

int main() {
    char str[100];

    printf("Enter a numeric string: ");
    scanf("%[^\n]s", str);

    int result = stringToInt(str);

    if (result != -1) {
        printf("Converted integer: %d\n", result);
    } else {
        printf("Invalid input! Please enter a valid numeric string.\n");
    }
```

```
    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>

void sortString(char str[]) {
    int n = strlen(str);
    char temp;

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (str[i] > str[j]) {
                temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }
}

int main() {
```

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

    printf("Enter first string: ");
    scanf("%[^\n]s", str1);
    getchar();  // Consume newline character
    printf("Enter second string: ");
    scanf("%[^\n]s", str2);

    // Convert both strings to lowercase and sort them
    for (int i = 0; str1[i]; i++) str1[i] = tolower(str1[i]);
    for (int i = 0; str2[i]; i++) str2[i] = tolower(str2[i]);

    sortString(str1);
    sortString(str2);

    if (strcmp(str1, str2) == 0) {
        printf("Strings are anagrams\n");
    } else {
        printf("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;

    while (str1[i] != '\0' && str2[j] != '\0') {
        result[k++] = str1[i++];
        result[k++] = str2[j++];
    }

    // Append remaining characters from str1 or str2
    while (str1[i] != '\0') {
        result[k++] = str1[i++];
    }
    while (str2[j] != '\0') {
        result[k++] = str2[j++];
    }

    result[k] = '\0';  // Null-terminate the merged string
}

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

    printf("Enter first string: ");
    scanf("%[^\n]s", str1);
```

```
    getchar();  // Consume newline character
    printf("Enter second string: ");
    scanf("%[^\n]s", str2);

    mergeStringsAlternately(str1, str2, result);

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

int countConsonants(char str[]) {
    int count = 0;

    for (int i = 0; str[i] != '\0'; i++) {
        char ch = tolower(str[i]);
        if ((ch >= 'a' && ch <= 'z') && !strchr("aeiou", ch)) {
            count++;
        }
```

```
    }

    return count;
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%[^\n]s", str);

    int consonantCount = countConsonants(str);

    printf("Number of consonants: %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>

void replaceSubstring(char str[], const char oldSub[], const char newSub[]) {
    char result[200];
```

```
    int i = 0, j = 0, k = 0;
    int oldLen = strlen(oldSub);

    while (str[i] != '\0') {
        if (strncmp(&str[i], oldSub, oldLen) == 0) {
            strcpy(&result[k], newSub);
            k += strlen(newSub);
            i += oldLen;
        } else {
            result[k++] = str[i++];
        }
    }

    result[k] = '\0';
    strcpy(str, result);
}

int main() {
    char str[200], oldSub[100], newSub[100];

    printf("Enter main string: ");
    scanf("%[^\n]s", str);
    getchar();  // Consume newline character
    printf("Enter substring to replace: ");
    scanf("%[^\n]s", oldSub);
    getchar();  // Consume newline character
    printf("Enter replacement substring: ");
    scanf("%[^\n]s", newSub);
```

```
    replaceSubstring(str, oldSub, newSub);

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

    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 countSubstringOccurrences(char str[], const char sub[]) {
    int count = 0;
    int subLen = strlen(sub);

    for (int i = 0; str[i] != '\0'; i++) {
        if (strncmp(&str[i], sub, subLen) == 0) {
            count++;
        }
    }

    return count;
}
```

```
int main() {
    char str[200], sub[100];

    printf("Enter main string: ");
    scanf("%[^\n]s", str);
    getchar();  // Consume newline character
    printf("Enter substring: ");
    scanf("%[^\n]s", sub);

    int count = countSubstringOccurrences(str, sub);

    printf("The substring appears %d times\n", count);

    return 0;
}
```

30.Implement 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>

int customStrlen(char str[]) {
    int length = 0;

    while (str[length] != '\0') {
        length++;
```

```
    }

    return length;
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%[^\n]s", str);

    int length = customStrlen(str);

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

    return 0;
}
```