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>
#include <math.h>

void computeStats(const double *array, int size, double *average, double *variance);



int main()
{
   int size;

   printf("enter the no of experiments : ");
   scanf("%d",&size);


   double array[size];
   double average, variance;

   printf("enter the elemnts :\n");
   for(int i=0;i<size;i++){
```

```
        printf("elemnt %d: ",i+1);
        scanf("%lf",&array[i]);
    }

     computeStats(array, size, &average, &variance);


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


    return 0;
}

// Function to compute the average and variance of an array
void computeStats(const double *array, int size, double *average, double *variance) {
    double sum = 0.0;
    double sum_of_squares = 0.0;


    for (int i = 0; i < size; i++) {
        sum += array[i];
        sum_of_squares += array[i] * array[i];
    }

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

    // Compute the variance
    *variance = (sum_of_squares / 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>

double* normalizeData(const double *array, int size);



int main() {
   int size;
   printf("enter the no of datas");
   scanf("%d",&size);

   double data[10];
   printf("enter the datas \n");
   for(int i=0;i<size;i++){
      printf(" data %d :",i+1);
      scanf("%lf",&data[i]);
   }



   double *normalizedData = normalizeData(data, size);

   if (normalizedData) {
```

```
        printf("Normalized Data:\n");
        for (int i = 0; i < size; i++) {
            printf("%.2f ", normalizedData[i]);
        }
        printf("\n");

    } else {
        printf("Normalization failed or not needed.\n");
    }

    return 0;
}

double* normalizeData(const double *array, int size) {
    if (size <= 0) return NULL;



    double min = array[0], max = array[0];
    for (int i = 1; i < size; i++) {
        if (array[i] < min) min = array[i];
        if (array[i] > max) max = array[i];
    }



    if (min == max) return NULL;



    double *normalizedArray = (double*)malloc(size * sizeof(double));
    if (!normalizedArray) {
        perror("Memory allocation failed");
        return NULL;
```

```
    }

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

    return normalizedArray;
}
```

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>

// Function prototype
void generateReport(const double *results, const char *descriptions[], int size);

// Function to generate a report summarizing experimental results and their descriptions
void generateReport(const double *results, const char *descriptions[], int size) {
    printf("Experimental Report:\n");
    printf("-----------------------\n");
    for (int i = 0; i < size; i++) {
        printf("Result %d:\n", i + 1);
        printf("Description: %s\n", descriptions[i]);
        printf("Value: %.2f\n", results[i]);
```

```
        printf("------------------------\n");
    }
}

int main() {
    int size;

    // Input the number of experimental results
    printf("Enter the number of experimental results: ");
    scanf("%d", &size);

    double results[size];
    char descriptions[size][100];

    // Input the experimental results and their descriptions
    printf("Enter the experimental results and their descriptions:\n");
    for (int i = 0; i < size; i++) {
        printf("Description for result %d: ", i + 1);
        scanf(" %[^\n]s", descriptions[i]);

        printf("Value for result %d: ", i + 1);
        scanf("%lf", &results[i]);
    }

    // Create array of pointers to descriptions
    const char *descriptionPointers[size];
    for (int i = 0; i < size; i++) {
        descriptionPointers[i] = descriptions[i];
    }

    // Generate the report
```

```
    generateReport(results, descriptionPointers, 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>

// Function prototype
void detectAnomalies(const double *data, int size, double threshold, int *anomalyCount);

// Function to detect anomalies in a dataset based on a threshold
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)++;
            printf("Anomaly detected at index %d: Value = %.2f\n", i, data[i]);
        }
    }
}

int main() {
```

```
    int size, anomalyCount;
    double threshold;

    // Input the size of the dataset
    printf("Enter the size of the dataset: ");
    scanf("%d", &size);

    double data[size];

    // Input the data points
    printf("Enter the data points:\n");
    for (int i = 0; i < size; i++) {
        printf("Data point %d: ", i + 1);
        scanf("%lf", &data[i]);
    }

    // Input the threshold for anomaly detection
    printf("Enter the threshold for anomaly detection: ");
    scanf("%lf", &threshold);

    // Detect anomalies in the dataset
    detectAnomalies(data, size, threshold, &anomalyCount);

    // Print the number of anomalies detected
    printf("Total number of anomalies detected: %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);

int main() {
    int size;
    double threshold;

    // Input the size of the dataset
    printf("Enter the size of the dataset: ");
    scanf("%d", &size);

    double data[size];
    char *labels[size];
```

```
    // Input the data points
    printf("Enter the data points:\n");
    for (int i = 0; i < size; i++) {
        printf("Data point %d: ", i + 1);
        scanf("%lf", &data[i]);
    }

    // Input the threshold for classification
    printf("Enter the threshold for classification: ");
    scanf("%lf", &threshold);

    // Classify the data points
    classifyData(data, size, labels, threshold);



    printf("Classification results:\n");
    for (int i = 0; i < size; i++) {
        printf("Data point %d: %.2f - %s\n", i + 1, data[i], labels[i]);
    }

    return 0;
}

// Function to classify data points into categories based on a threshold
void classifyData(const double *data, int size, char *labels[], double threshold) {
    for (int i = 0; i < size; i++) {
        if (data[i] > threshold) {
            labels[i] = "Above Threshold";
        } else {
            labels[i] = "Below Threshold";
        }
```

```
    }
}
```

Artificial Intelligence

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 prototype
void adjustWeights(double *weights, int size, double learningRate);

// Function to adjust neural network weights using a given learning rate
void adjustWeights(double *weights, int size, double learningRate) {
    for (int i = 0; i < size; i++) {
        // Example adjustment: increment weight by learning rate
        weights[i] += learningRate * weights[i];
    }
}

int main() {
    int size;
    double learningRate;
```

```
    // Input the number of weights
    printf("Enter the number of weights: ");
    scanf("%d", &size);

    double weights[size];

    // Input the weight values
    printf("Enter the weight values:\n");
    for (int i = 0; i < size; i++) {
        printf("Weight %d: ", i + 1);
        scanf("%lf", &weights[i]);
    }

    // Input the learning rate
    printf("Enter the learning rate: ");
    scanf("%lf", &learningRate);

    // Adjust the weights
    adjustWeights(weights, size, learningRate);

    // Print the adjusted weights
    printf("Adjusted weights:\n");
    for (int i = 0; i < size; i++) {
        printf("Weight %d: %.2f\n", i + 1, 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);

// Function to evaluate multiple AI models and determine the best accuracy
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() {
    int size;
    double bestAccuracy;
```

```
    printf("Enter the number of AI models: ");
    scanf("%d", &size);

    double accuracies[size];

    printf("Enter the accuracy values for each AI model:\n");
    for (int i = 0; i < size; i++) {
        printf("Accuracy of model %d: ", i + 1);
        scanf("%lf", &accuracies[i]);
    }
    evaluateModels(accuracies, size, &bestAccuracy);

    // Print the best accuracy
    printf("The best accuracy among the AI models 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>
void constructDecisionTree(const double *features, int size, int *treeStructure);
```

```
// Function
void constructDecisionTree(const double *features, int size, int *treeStructure) {

    for (int i = 0; i < size; i++) {
        if (features[i] < 0.5) {
            treeStructure[i] = 0;
        } else {
            treeStructure[i] = 1;
        }
    }
}

int main() {
    int size;

    printf("Enter the number of features: ");
    scanf("%d", &size);

    double features[size];
    int treeStructure[size];
    printf("Enter the feature data (values between 0 and 1):\n");
    for (int i = 0; i < size; i++) {
        printf("Feature %d: ", i + 1);
        scanf("%lf", &features[i]);
    }


    constructDecisionTree(features, size, treeStructure);

    printf("Decision Tree Structure:\n");
```

```
    for (int i = 0; i < size; i++) {
        printf("Node %d: %s\n", i + 1, (treeStructure[i] == 0) ? "Left child" : "Right child");
    }

    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) {
    // Predefined positive and negative words
    const char *positiveWords[] = {"good", "great", "happy", "excellent", "amazing", "positive", "joy"};
    const char *negativeWords[] = {"bad", "sad", "terrible", "horrible", "negative", "angry", "pain"};
    int positiveCount = sizeof(positiveWords) / sizeof(positiveWords[0]);
    int negativeCount = sizeof(negativeWords) / sizeof(negativeWords[0]);

    for (int i = 0; i < size; i++) {
        int score = 0; // Reset sentiment score for the current sentence
        const char *sentence = sentences[i];
```

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

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

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

int main() {

    const char *sentences[] = {
        "I am feeling very good today!",
        "This is a bad experience.",
        "What a great and amazing day!",
        "I am sad and angry.",
        "This product is excellent, truly positive!",
        "Horrible service, very bad."
    };
```

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

    // Process sentiment analysis
    processSentiments(sentences, size, sentimentScores);

    // Print the results
    printf("Sentiment Scores:\n");
    for (int i = 0; i < size; i++) {
        printf("Sentence: \"%s\"\nScore: %d\n\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>
double* generateTrainingData(const double *baseData, int size, int multiplier);

// Function to generate training data by applying a multiplier to base data
double* generateTrainingData(const double *baseData, int size, int multiplier) {
    static double trainingData[100]; // Assumes the size won't exceed 100
```

```
    if (size > 100) {
        printf("Size exceeds the limit.\n");
        return NULL;
    }

    for (int i = 0; i < size; i++) {
        trainingData[i] = baseData[i] * multiplier;
    }

    return trainingData;
}

int main() {
    int size, multiplier;

    printf("Enter the size of the base data (up to 100): ");
    scanf("%d", &size);

    if (size > 100) {
        printf("Size exceeds the limit.\n");
        return 1;
    }

    double baseData[size];

    printf("Enter the base data values:\n");
    for (int i = 0; i < size; i++) {
        printf("Base data %d: ", i + 1);
        scanf("%lf", &baseData[i]);
    }
```

```
    printf("Enter the multiplier: ");
    scanf("%d", &multiplier);



    double *trainingData = generateTrainingData(baseData, size, multiplier);

    if (trainingData != NULL) {

        printf("Training data:\n");
        for (int i = 0; i < size; i++) {
            printf("Training data %d: %.2f\n", i + 1, trainingData[i]);
        }
    }

    return 0;
}
```

Computer Vision

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>
```

```
double* generateTrainingData(const double *baseData, int size, int multiplier);

// Function to generate training data by applying a multiplier to base data
double* generateTrainingData(const double *baseData, int size, int multiplier) {
    static double trainingData[100]; // Assumes the size won't exceed 100

    if (size > 100) {
        printf("Size exceeds the limit.\n");
        return NULL;
    }

    for (int i = 0; i < size; i++) {
        trainingData[i] = baseData[i] * multiplier;
    }

    return trainingData;
}

int main() {
    int size, multiplier;


    printf("Enter the size of the base data (up to 100): ");
    scanf("%d", &size);

    if (size > 100) {
        printf("Size exceeds the limit.\n");
        return 1;
    }
```

```
    double baseData[size];


    printf("Enter the base data values:\n");
    for (int i = 0; i < size; i++) {
        printf("Base data %d: ", i + 1);
        scanf("%lf", &baseData[i]);
    }

    // Input the multiplier
    printf("Enter the multiplier: ");
    scanf("%d", &multiplier);


    double *trainingData = generateTrainingData(baseData, size, multiplier);

    if (trainingData != NULL) {

        printf("Training data:\n");
        for (int i = 0; i < size; i++) {
            printf("Training data %d: %.2f\n", i + 1, trainingData[i]);
        }
    }

    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>

#define WIDTH  5
#define HEIGHT 5

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

// Function to apply an averaging filter
void applyFilter(const unsigned char *image, unsigned char *filteredImage, int width, int height) {
   for (int y = 1; y < height - 1; y++) {
      for (int x = 1; x < width - 1; x++) {
         int sum = 0;
         for (int dy = -1; dy <= 1; dy++) {
            for (int dx = -1; dx <= 1; dx++) {
               sum += image[(y + dy) * width + (x + dx)];
            }
         }
         filteredImage[y * width + x] = sum / 9;
      }
   }
}

int main() {
   unsigned char image[WIDTH * HEIGHT] = {
      1, 2, 3, 4, 5,
```

```
    6, 7, 8, 9, 10,
    11, 12, 13, 14, 15,
    16, 17, 18, 19, 20,
    21, 22, 23, 24, 25
  };

  unsigned char filteredImage[WIDTH * HEIGHT] = {0};

  applyFilter(image, filteredImage, WIDTH, HEIGHT);



  printf("Filtered Image:\n");
  for (int y = 0; y < HEIGHT; y++) {
    for (int x = 0; x < WIDTH; x++) {
      printf("%3d ", filteredImage[y * WIDTH + x]);
    }
    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>

// Function to recognize objects based on features
void recognizeObjects(const double *features, int size, char *objectLabels[]) {
    // Decision-making rules for object recognition
    for (int i = 0; i < size; i++) {
        if (features[i] < 1.0) {
            objectLabels[i] = "Sphere"; // Example object
        } else if (features[i] >= 1.0 && features[i] < 5.0) {
            objectLabels[i] = "Cube"; // Example object
        } else if (features[i] >= 5.0 && features[i] < 10.0) {
            objectLabels[i] = "Pyramid"; // Example object
        } else {
            objectLabels[i] = "Unknown"; // Fallback label
        }
    }
}

int main() {
    // Example feature vectors
    double features[] = {0.5, 3.2, 7.8, 12.4, 4.5};
    int size = sizeof(features) / sizeof(features[0]);

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


    recognizeObjects(features, size, objectLabels);
```

```
    printf("Recognized Objects:\n");
    for (int i = 0; i < size; i++) {
        printf("Feature: %.2f -> Object: %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>

void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth, int originalHeight, int newWidth, int newHeight);

// Function to resize an image using bilinear interpolation
void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth, int originalHeight, int newWidth, int newHeight) {
    for (int y = 0; y < newHeight; y++) {
        for (int x = 0; x < newWidth; x++) {
```

```
            float gx = ((float)x / newWidth) * (originalWidth - 1);
            float gy = ((float)y / newHeight) * (originalHeight - 1);
            int gxi = (int)gx;
            int gyi = (int)gy;
            int gxi1 = gxi + 1;
            int gyi1 = gyi + 1;

            unsigned char c00 = inputImage[gyi * originalWidth + gxi];
            unsigned char c10 = inputImage[gyi * originalWidth + gxi1];
            unsigned char c01 = inputImage[gyi1 * originalWidth + gxi];
            unsigned char c11 = inputImage[gyi1 * originalWidth + gxi1];

            float tx = gx - gxi;
            float ty = gy - gyi;
            unsigned char c = (unsigned char)((c00 * (1 - tx) * (1 - ty)) + (c10 * tx * (1 - ty)) + (c01 * (1 - tx)
* ty) + (c11 * tx * ty));

            outputImage[y * newWidth + x] = c;
        }
    }
}

int main() {
    int originalWidth = 4;
    int originalHeight = 4;
    unsigned char inputImage[16] = {
        10, 20, 30, 40,
        50, 60, 70, 80,
        90, 100, 110, 120,
        130, 140, 150, 160
    };
```

```
    int newWidth = 8;
    int newHeight = 8;
    unsigned char outputImage[64] = {0};


    resizeImage(inputImage, outputImage, originalWidth, originalHeight, newWidth, newHeight);


    printf("Resized Image:\n");
    for (int y = 0; y < newHeight; y++) {
        for (int x = 0; x < newWidth; x++) {
            printf("%3d ", outputImage[y * newWidth + x]);
        }
        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>

#define WIDTH  5
#define HEIGHT 5

void balanceColors(const unsigned char *image, unsigned char *balancedImage, int width, int height);

// Function to adjust the color balance of an image
void balanceColors(const unsigned char *image, unsigned char *balancedImage, int width, int height) {
   const float brightnessFactor = 1.2; // Example factor to increase brightness
   const float contrastFactor = 1.1;   // Example factor to increase contrast

   for (int y = 0; y < height; y++) {
      for (int x = 0; x < width; x++) {
         int index = y * width + x;
         float pixel = image[index];

         // Adjust brightness
         pixel *= brightnessFactor;

         // Adjust contrast
         pixel = ((pixel - 128) * contrastFactor) + 128;

         // Ensure the pixel value stays within valid range
         if (pixel > 255) pixel = 255;
         if (pixel < 0) pixel = 0;

         balancedImage[index] = (unsigned char)pixel;
      }
   }
```

```
}

int main() {
    unsigned char image[WIDTH * HEIGHT] = {
        10, 20, 30, 40, 50,
        60, 70, 80, 90, 100,
        110, 120, 130, 140, 150,
        160, 170, 180, 190, 200,
        210, 220, 230, 240, 250
    };

    unsigned char balancedImage[WIDTH * HEIGHT] = {0};

    // Adjust the color balance of the image
    balanceColors(image, balancedImage, WIDTH, HEIGHT);

    // Print the balanced image
    printf("Balanced Image:\n");
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            printf("%3d ", balancedImage[y * WIDTH + x]);
        }
        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>

// Function to recognize patterns in a dataset
void recognizePatterns(const char *patterns[], int size, int *matchCounts) {

    const char *dataset[] = {
        "apple", "banana", "cherry", "apple", "date",
        "banana", "apple", "fig", "grape", "cherry"
    };
    int datasetSize = sizeof(dataset) / sizeof(dataset[0]);

    // Initialize match counts to 0
    for (int i = 0; i < size; i++) {
        matchCounts[i] = 0;
    }


    for (int i = 0; i < size; i++) {
        for (int j = 0; j < datasetSize; j++) {
            if (strcmp(patterns[i], dataset[j]) == 0) {
                matchCounts[i]++;
            }
```

```
        }
    }
}

int main() {
    // Patterns to recognize
    const char *patterns[] = {"apple", "banana", "cherry"};
    int size = sizeof(patterns) / sizeof(patterns[0]);

    // Array to store match counts
    int matchCounts[size];

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

    // Output match counts
    printf("Pattern Recognition Results:\n");
    for (int i = 0; i < size; i++) {
        printf("Pattern '%s': %d matches\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>

// Function prototype
void analyzeClimateData(const double *temperatureReadings, int size, double *minTemp, double *maxTemp);

// Function to analyze climate data and find min and max temperatures
void analyzeClimateData(const double *temperatureReadings, int size, double *minTemp, double *maxTemp) {
    *minTemp = temperatureReadings[0];
    *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() {
    int size;

    // Input the number of temperature readings
```

```
    printf("Enter the number of temperature readings: ");
    scanf("%d", &size);

    double temperatureReadings[size];
    double minTemp, maxTemp;

    // Input the temperature readings
    printf("Enter the temperature readings:\n");
    for (int i = 0; i < size; i++) {
        printf("Reading %d: ", i + 1);
        scanf("%lf", &temperatureReadings[i]);
    }

    // Analyze the climate data to find min and max temperatures
    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);

// Function to process quantum measurement data
void processQuantumData(const double *measurements, int size, double *processedData) {
    for (int i = 0; i < size; i++) {
        // Example processing: square the measurement values
        processedData[i] = measurements[i] * measurements[i];
    }
}

int main() {
    int size;

    // Input the number of measurements
    printf("Enter the number of measurements: ");
    scanf("%d", &size);

    double measurements[size];
    double processedData[size];

    // Input the measurement values
    printf("Enter the measurement values:\n");
    for (int i = 0; i < size; i++) {
        printf("Measurement %d: ", i + 1);
```

```
        scanf("%lf", &measurements[i]);
    }

    // Process the quantum measurement data
    processQuantumData(measurements, size, processedData);

    // Print the processed data
    printf("Processed Data:\n");
    for (int i = 0; i < size; i++) {
        printf("Processed data %d: %.2f\n", i + 1, processedData[i]);
    }

    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);

// Function to visualize scientific data with a given title
void visualizeData(const double *data, int size, const char *title) {
    printf("%s\n", title);
    printf("----------------------------\n");

    for (int i = 0; i < size; i++) {
        printf("Data %d: ", i + 1);
        int barLength = (int)(data[i] * 50); // Scale the data for visualization
        for (int j = 0; j < barLength; j++) {
            printf("*");
        }
        printf(" (%.2f)\n", data[i]);
    }
    printf("----------------------------\n");
}

int main() {
    int size;

    printf("Enter the number of data points: ");
    scanf("%d", &size);

    double data[size];
    char title[100];

    printf("Enter the data points:\n");
```

```
    for (int i = 0; i < size; i++) {
        printf("Data point %d: ", i + 1);
        scanf("%lf", &data[i]);
    }

    printf("Enter the title for the visualization: ");
    scanf(" %[^\n]s", title);
    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>

double* simulateGeneticData(const double *initialData, int size, double mutationRate);

// Function to simulate genetic data evolution by applying a mutation rate
double* simulateGeneticData(const double *initialData, int size, double mutationRate) {
    static double simulatedData[100]; // Assumes the size won't exceed 100

    if (size > 100) {
        printf("Size exceeds the limit.\n");
```

```
    return NULL;
  }

  // Static mutation factors for demonstration
  double mutationFactors[] = {0.1, -0.2, 0.3, -0.1, 0.05};

  for (int i = 0; i < size; i++) {
    double mutation = mutationFactors[i % 5] * mutationRate;
    simulatedData[i] = initialData[i] + mutation;
  }

  return simulatedData;
}

int main() {
  int size;
  double mutationRate;

  // Input the size of the initial data
  printf("Enter the size of the initial data (up to 100): ");
  scanf("%d", &size);

  if (size > 100) {
    printf("Size exceeds the limit.\n");
    return 1;
  }

  double initialData[size];

  // Input the initial data values
  printf("Enter the initial data values:\n");
```

```
    for (int i = 0; i < size; i++) {
        printf("Initial data %d: ", i + 1);
        scanf("%lf", &initialData[i]);
    }

    // Input the mutation rate
    printf("Enter the mutation rate: ");
    scanf("%lf", &mutationRate);

    // Simulate the genetic data evolution
    double *simulatedData = simulateGeneticData(initialData, size, mutationRate);

    if (simulatedData != NULL) {
        // Print the simulated data
        printf("Simulated Data:\n");
        for (int i = 0; i < size; i++) {
            printf("Simulated data %d: %.2f\n", i + 1, simulatedData[i]);
        }
    }

    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>

// Function prototype
void trackPerformance(const double *performanceData, int size, double *maxPerformance, double *minPerformance);

// Function to track AI performance data
void trackPerformance(const double *performanceData, int size, double *maxPerformance, double *minPerformance) {
    *maxPerformance = performanceData[0];
    *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() {
    int size;
    double maxPerformance, minPerformance;

    printf("Enter the number of performance data points: ");
    scanf("%d", &size);

    double performanceData[size];
```

```
    printf("Enter the performance data points:\n");
    for (int i = 0; i < size; i++) {
        printf("Performance data %d: ", i + 1);
        scanf("%lf", &performanceData[i]);
    }

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

    printf("Maximum Performance: %.2f\n", maxPerformance);
    printf("Minimum 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>

// Function prototype
```

```
void filterSensorData(const double *sensorData, double *filteredData, int size, double filterThreshold);


// Function to filter sensor data based on a threshold
void filterSensorData(const double *sensorData, double *filteredData, int size, double filterThreshold) {
    for (int i = 0; i < size; i++) {
        if (sensorData[i] > filterThreshold) {
            filteredData[i] = sensorData[i];
        } else {
            filteredData[i] = 0;  // Example: Set to 0 if below the threshold
        }
    }
}

int main() {
    int size;
    double filterThreshold;

    // Input the number of sensor data points
    printf("Enter the number of sensor data points: ");
    scanf("%d", &size);

    double sensorData[size];
    double filteredData[size];

    // Input the sensor data points
    printf("Enter the sensor data points:\n");
    for (int i = 0; i < size; i++) {
        printf("Sensor data %d: ", i + 1);
        scanf("%lf", &sensorData[i]);
    }
```

```
    // Input the filter threshold
    printf("Enter the filter threshold: ");
    scanf("%lf", &filterThreshold);

    // Filter the sensor data
    filterSensorData(sensorData, filteredData, size, filterThreshold);

    // Print the filtered data
    printf("Filtered Data:\n");
    for (int i = 0; i < size; i++) {
        printf("Filtered data %d: %.2f\n", i + 1, 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>
```

```
// Function prototype
void planLogistics(const double *resourceLevels, double *logisticsPlan, int size);

// Function to plan logistics based on resource levels
void planLogistics(const double *resourceLevels, double *logisticsPlan, int size) {
    double totalResources = 0.0;
    for (int i = 0; i < size; i++) {
        totalResources += resourceLevels[i];
    }

    double averageResources = totalResources / size;

    for (int i = 0; i < size; i++) {
        // Simple example logic: if resource level is above average, allocate less; otherwise, allocate more
        if (resourceLevels[i] > averageResources) {
            logisticsPlan[i] = resourceLevels[i] * 0.75; // Allocate 75% if above average
        } else {
            logisticsPlan[i] = resourceLevels[i] * 1.25; // Allocate 125% if below average
        }
    }
}

int main() {
    int size;

    // Input the number of resource levels
    printf("Enter the number of resource levels: ");
    scanf("%d", &size);

    double resourceLevels[size];
```

```
    double logisticsPlan[size];

    // Input the resource levels
    printf("Enter the resource levels:\n");
    for (int i = 0; i < size; i++) {
        printf("Resource level %d: ", i + 1);
        scanf("%lf", &resourceLevels[i]);
    }

    // Plan logistics based on resource levels
    planLogistics(resourceLevels, logisticsPlan, size);

    // Print the logistics plan
    printf("Logistics Plan:\n");
    for (int i = 0; i < size; i++) {
        printf("Logistics plan %d: %.2f\n", i + 1, logisticsPlan[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
void processSatelliteImage(const unsigned char *imageData, unsigned char *processedImage, int width, int height);

// Function to process satellite image data (example: edge detection)
void processSatelliteImage(const unsigned char *imageData, unsigned char *processedImage, int width, int height) {
    // Example edge detection using simple gradient-based technique
    for (int y = 1; y < height - 1; y++) {
        for (int x = 1; x < width - 1; x++) {
            int gx = -imageData[(y - 1) * width + (x - 1)] + imageData[(y - 1) * width + (x + 1)]
                     -2 * imageData[y * width + (x - 1)] + 2 * imageData[y * width + (x + 1)]
                     -imageData[(y + 1) * width + (x - 1)] + imageData[(y + 1) * width + (x + 1)];

            int gy = -imageData[(y - 1) * width + (x - 1)] - 2 * imageData[(y - 1) * width + x] - imageData[(y - 1) * width + (x + 1)]
                     +imageData[(y + 1) * width + (x - 1)] + 2 * imageData[(y + 1) * width + x] + imageData[(y + 1) * width + (x + 1)];

            int magnitude = (int)sqrt((gx * gx) + (gy * gy));
            if (magnitude > 255) {
                magnitude = 255;
            }

            processedImage[y * width + x] = (unsigned char)magnitude;
        }
    }
}

int main() {
    int width = 5;
```

```
    int height = 5;
    unsigned char imageData[25] = {
        10, 20, 30, 40, 50,
        60, 70, 80, 90, 100,
        110, 120, 130, 140, 150,
        160, 170, 180, 190, 200,
        210, 220, 230, 240, 250
    };

    unsigned char processedImage[25] = {0};

    // Process the satellite image data
    processSatelliteImage(imageData, processedImage, width, height);

    // Print the processed image data
    printf("Processed Image:\n");
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            printf("%3d ", processedImage[y * width + x]);
        }
        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);

// Function to analyze and optimize flight path coordinates
void analyzeFlightPath(const double *pathCoordinates, double *optimizedPath, int size) {
    // Example smoothing algorithm: Moving average
    for (int i = 1; i < size - 1; i++) {
        optimizedPath[i] = (pathCoordinates[i - 1] + pathCoordinates[i] + pathCoordinates[i + 1]) / 3.0;
    }

    optimizedPath[0] = pathCoordinates[0];
    optimizedPath[size - 1] = pathCoordinates[size - 1];
}

int main() {
    int size;



    printf("Enter the number of path coordinates: ");
    scanf("%d", &size);

    double pathCoordinates[size];
    double optimizedPath[size];
```

```
    printf("Enter the path coordinates:\n");
    for (int i = 0; i < size; i++) {
        printf("Coordinate %d: ", i + 1);
        scanf("%lf", &pathCoordinates[i]);
    }

    // Analyze and optimize the flight path coordinates
    analyzeFlightPath(pathCoordinates, optimizedPath, size);



    printf("Optimized Path:\n");
    for (int i = 0; i < size; i++) {
        printf("Coordinate %d: %.2f\n", i + 1, optimizedPath[i]);
    }

    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);


// Function to augment AI data by applying an augmentation factor
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() {
   int size;
   double augmentationFactor;



   printf("Enter the size of the original data: ");
   scanf("%d", &size);

   double originalData[size];
   double augmentedData[size];



   printf("Enter the original data values:\n");
   for (int i = 0; i < size; i++) {
      printf("Original data %d: ", i + 1);
      scanf("%lf", &originalData[i]);
   }

   printf("Enter the augmentation factor: ");
   scanf("%lf", &augmentationFactor);
```

```
    augmentData(originalData, augmentedData, size, augmentationFactor);

    printf("Augmented Data:\n");
    for (int i = 0; i < size; i++) {
        printf("Augmented data %d: %.2f\n", i + 1, augmentedData[i]);
    }

    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);

// Function to analyze medical image data using thresholding
void analyzeMedicalImage(const unsigned char *imageData, unsigned char *analysisResults, int width, int height) {
    const unsigned char threshold = 128; // Example threshold value
```

```
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int index = y * width + x;
            if (imageData[index] > threshold) {
                analysisResults[index] = 255; // Above threshold, set to white
            } else {
                analysisResults[index] = 0;   // Below threshold, set to black
            }
        }
    }
}

int main() {
    int width = 5;
    int height = 5;
    unsigned char imageData[25] = {
        10, 50, 130, 200, 250,
        20, 60, 140, 210, 240,
        30, 70, 150, 220, 230,
        40, 80, 160, 230, 220,
        50, 90, 170, 240, 210
    };

    unsigned char analysisResults[25] = {0};

    // Analyze the medical image data
    analyzeMedicalImage(imageData, analysisResults, width, height);

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

// Function to track objects based on data
void trackObjects(const double *objectData, double *trackingResults, int size) {
    for (int i = 0; i < size; i++) {
        // Simple example logic: apply a constant velocity to update positions
```

```
        // For demonstration, assuming velocity is 1.0 for all objects
        double velocity = 1.0;
        trackingResults[i] = objectData[i] + velocity;
    }
}

int main() {
    int size;

    // Input the number of objects
    printf("Enter the number of objects: ");
    scanf("%d", &size);

    double objectData[size];
    double trackingResults[size];

    // Input the object data (initial positions)
    printf("Enter the initial positions of the objects:\n");
    for (int i = 0; i < size; i++) {
        printf("Object %d initial position: ", i + 1);
        scanf("%lf", &objectData[i]);
    }

    trackObjects(objectData, trackingResults, size);
    printf("Tracking Results (New Positions):\n");
    for (int i = 0; i < size; i++) {
        printf("Object %d new position: %.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>

void optimizeDefenseStrategy(const double *threatLevels, double *optimizedStrategies, int size);

// Function to optimize defense strategies based on threat levels
void optimizeDefenseStrategy(const double *threatLevels, double *optimizedStrategies, int size) {
    double totalThreat = 0.0;
    for (int i = 0; i < size; i++) {
        totalThreat += threatLevels[i];
    }

    double averageThreat = totalThreat / size;

    for (int i = 0; i < size; i++) {
        // Simple example logic: if threat level is above average, allocate more resources; otherwise, allocate fewer resources
        if (threatLevels[i] > averageThreat) {
            optimizedStrategies[i] = threatLevels[i] * 1.25; // Allocate 125% if above average
        } else {
            optimizedStrategies[i] = threatLevels[i] * 0.75; // Allocate 75% if below average
        }
    }
}
```

```
int main() {
    int size;

    printf("Enter the number of threat levels: ");
    scanf("%d", &size);

    double threatLevels[size];
    double optimizedStrategies[size];

    printf("Enter the threat levels:\n");
    for (int i = 0; i < size; i++) {
        printf("Threat level %d: ", i + 1);
        scanf("%lf", &threatLevels[i]);
    }

    optimizeDefenseStrategy(threatLevels, optimizedStrategies, size);

    printf("Optimized Defense Strategies:\n");
    for (int i = 0; i < size; i++) {
        printf("Strategy %d: %.2f\n", i + 1, optimizedStrategies[i]);
    }

    return 0;
}
```

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];



   printf("Enter a string: ");
   scanf("%99[^\n]", str);



   size_t length = strlen(str);



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

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 source[100];
   char destination[50];

   printf("Enter the source string: ");
   scanf("%99s", source);

   // Validate if the source string fits into the destination buffer
   if (strlen(source) >= sizeof(destination)) {
      printf("Error: The source string is too long to fit into the destination buffer.\n");
   } else {
      // Copy the source string to the destination string using strcpy()
      strcpy(destination, source);
      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[200];
    char str2[100];

    // Input the first string
    printf("Enter the first string: ");
    scanf("%199s", str1);

    // Input the second string
    printf("Enter the second string: ");
    scanf("%99s", str2);

    // Check if the destination string has enough space to hold the result
    if (strlen(str1) + strlen(str2) >= sizeof(str1)) {
        printf("Error: The combined string is too long to fit into the destination buffer.\n");
    } else {
        // Concatenate the strings using strcat()
        strcat(str1, str2);
        printf("Concatenated string: %s\n", str1);
    }

    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];
    char str2[100];

    // Input the first string
    printf("Enter the first string: ");
    scanf("%99s", str1);

    // Input the second string
    printf("Enter the second string: ");
    scanf("%99s", str2);

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

    // Output the comparison result
    if (result < 0) {
        printf("'%s' is less than '%s'\n", str1, str2);
    } else if (result > 0) {
        printf("'%s' is greater than '%s'\n", str1, str2);
```

```
    } else {
        printf("'%s' is equal to '%s'\n", str1, str2);
    }

    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>


void convertToUppercase(char *str) {
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] >= 'a' && str[i] <= 'z') {
            str[i] = str[i] - ('a' - 'A');
        }
    }
}

int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%99[^\n]", str);

    // Convert the string to uppercase
    convertToUppercase(str);
```

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

    return 0;
}
```

7. 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>

// Custom implementation to convert a string to lowercase
void convertToLowercase(char *str) {
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] >= 'A' && str[i] <= 'Z') {
            str[i] = str[i] + ('a' - 'A');
        }
    }
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%99[^\n]", str);
```

```
    convertToLowercase(str);

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

    return 0;

}
```

8. 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 *pos;

    // Input main string
    printf("Enter the main string: ");
    scanf("%99[^\n]", mainStr);

    getchar();

    printf("Enter the substring to search: ");
    scanf("%99[^\n]", subStr);
```

```
    pos = strstr(mainStr, subStr);

    if (pos) {

        int index = pos - mainStr;
        printf("Substring found at index: %d\n", index);
    } else {
        printf("Substring not found in the main string.\n");
    }

    return 0;
}
```

9. 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 findCharacter(const char *str, char ch) {
    char *pos = strchr(str, ch);
    if (pos != NULL) {
        return pos - str;
    } else {
        return -1;
```

```
    }
}

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

    printf("Enter a string: ");
    scanf("%99[^\n]", str);

    int dummy;
    while ((dummy = getchar()) != '\n' && dummy != EOF);

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

    int index = findCharacter(str, ch);

    if (index != -1) {
        printf("The character '%c' is found at index: %d\n", ch, index);
    } else {
        printf("The character '%c' is not found.\n", ch);
    }

    return 0;
}
```

10. 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>

// Function to reverse a string in place
void reverseString(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        // Swap the characters at positions i and (len - i - 1)
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}

int main() {
    char str[100];

    // Input the string from the user
    printf("Enter a string: ");
    scanf("%99[^\n]", str);

    reverseString(str);
```

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

    return 0;
}
```

11. 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];
    char *token;
    int count = 0;

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

    token = strtok(input, " ");

    while (token != NULL) {
        count++;
        token = strtok(NULL, " ");
    }
```

```
    printf("Number of words (tokens): %d\n", count);

    return 0;
}
```

12. 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>
#include <stdlib.h>

int main() {
    char input[100];
    char *duplicatedString;

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

    duplicatedString = strdup(input);

    if (duplicatedString == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }
```

```
    printf("Original string: %s\n", input);
    printf("Duplicated string: %s\n", duplicatedString);




    return 0;
}
```

13. 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 string1[100], string2[100];
    int result;

    printf("Enter the first string: ");
    scanf("%[^\n]%*c", string1);

    printf("Enter the second string: ");
    scanf("%[^\n]", string2);



    result = strcasecmp(string1, string2);
```

```
    if (result == 0) {
        printf("The strings are equal (case-insensitive comparison).\n");
    } else {
        printf("The strings are different (case-insensitive comparison).\n");
    }

    return 0;
}
```

14. 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 trimWhitespace(char *str) {
    char *start, *end;
    int length;

    start = str;
    while (*start && (*start == ' ' || *start == '\t' || *start == '\n' || *start == '\r')) {
        start++;
    }
```

```
    end = str + strlen(str) - 1;
    while (end > start && (*end == ' ' || *end == '\t' || *end == '\n' || *end == '\r')) {
        end--;
    }

    length = end - start + 1;

    for (int i = 0; i < length; i++) {
        str[i] = start[i];
    }

    str[length] = '\0';
}

int main() {
    char input[100];

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

    printf("Trimmed string: '%s'\n", input);

    return 0;
}
```

15. 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>

// Function to find the last occurrence of a character in a string
int find_last_occurrence(char *str, char ch) {
    int last_index = -1;

    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            last_index = i;
        }
    }
    return last_index;
}

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

    printf("Enter a string: ");
    scanf("%99s", str);

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

```
    scanf(" %c", &ch);

    int result = find_last_occurrence(str, ch);

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

    return 0;
}
```

16. 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 count_vowels(char *str) {
    int count = 0;
    char ch;

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

```
        if (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ||
            ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            count++;
        }
    }

    return count;
}

int main() {
    char str[100];


    printf("Enter a string: ");
    scanf("%99s", str);


    int result = count_vowels(str);


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

    return 0;
}
```

17. 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>



char to_lower(char ch) {
    if (ch >= 'A' && ch <= 'Z') {
        return ch + ('a' - 'A');
    }
    return ch;
}



int count_specific_char(char *str, char ch, int case_sensitive) {
    int count = 0;



    if (!case_sensitive) {
        ch = to_lower(ch);
    }



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

```
        if (!case_sensitive) {
            current_char = to_lower(current_char);
        }


        if (current_char == ch) {
            count++;
        }
    }

    return count;
}

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


    printf("Enter a string: ");
    scanf("%99s", str);

    printf("Enter a character to count: ");
    scanf(" %c", &ch);
    printf("Should the count be case sensitive? (1 for Yes, 0 for No): ");
    scanf("%d", &case_sensitive);


    int result = count_specific_char(str, ch, case_sensitive);
```

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

    return 0;
}
```

18.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 removeCharacter(char *str, char charToRemove) {
    int i, j;
    for (i = 0, j = 0; str[i] != '\0'; i++) {
        if (str[i] != charToRemove) {
            str[j++] = str[i];
        }
    }
    str[j] = '\0'; // Terminate the modified string
}

int main() {
```

```
    char str[100];
    char charToRemove;

    // Input the string
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    // Remove the newline character if it exists
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n') {
        str[len - 1] = '\0';
    }

    // Input the character to remove
    printf("Enter the character to remove: ");
    scanf(" %c", &charToRemove);

    // Remove the character from the string
    removeCharacter(str, charToRemove);

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

    return 0;
}
```

19.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>

// Function to convert a character to lowercase
char toLower(char c) {
    if (c >= 'A' && c <= 'Z') {
        return c + ('a' - 'A');
    }
    return c;
}
int isAlnum(char c) {
    return (c >= 'A' && c <= 'Z') ||
           (c >= 'a' && c <= 'z') ||
           (c >= '0' && c <= '9');
}

int isPalindrome(const char *str) {
    int left = 0;
    int right = strlen(str) - 1;

    while (left < right) {
        // Skip non-alphanumeric characters
        while (left < right && !isAlnum(str[left])) left++;
        while (left < right && !isAlnum(str[right])) right--;

        // Compare characters, ignoring case
        if (toLower(str[left]) != toLower(str[right])) {
            return 0; // Not a palindrome
```

```
        }

        left++;
        right--;
    }

    return 1; // Is a palindrome
}

int main() {
    char str[100];

    // Input the string
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    // Remove the newline character if it exists
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n') {
        str[len - 1] = '\0';
    }

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

    return 0;
}
```

20.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(const char *mainString, char *substring, int startIndex, int length) {
    int mainStringLength = strlen(mainString);

    if (startIndex < 0 || startIndex >= mainStringLength || length < 0 || (startIndex + length) > mainStringLength) {
        printf("Error: Invalid start index or length.\n");
        substring[0] = '\0';
        return;
    }

    strncpy(substring, mainString + startIndex, length);
```

```
    substring[length] = '\0';
}

int main() {
    char mainString[100];
    int startIndex, length;
    char substring[100];

    // Input the main string using scanf
    printf("Enter the main string: ");
    scanf("%99[^\n]", mainString); // Read until newline, with a maximum length of 99 characters

    // Clear the input buffer
    int ch;
    while ((ch = getchar()) != '\n' && ch != EOF);

    // Input the start index and length using scanf
    printf("Enter the start index: ");
    scanf("%d", &startIndex);
    printf("Enter the length: ");
    scanf("%d", &length);


    extractSubstring(mainString, substring, startIndex, length);


    if (substring[0] != '\0') {
        printf("Extracted substring: %s\n", substring);
    }

    return 0;
```

```
}
```

21. 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 sortCharacters(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len - 1; i++) {
        for (int j = i + 1; j < len; j++) {
            if (str[i] > str[j]) {
                // Swap the characters
                char temp = str[i];
                str[i] = str[j];
                str[j] = temp;
            }
        }
    }
```

```
}

int main() {
    char str[100];


    printf("Enter a string: ");
    scanf("%s", str);


    sortCharacters(str);

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

    return 0;
}
```

22.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 countWords(char *str) {
    int wordCount = 0;
    char *token = strtok(str, " ");
```

```
    while (token != NULL) {
        wordCount++;
        token = strtok(NULL, " ");
    }

    return wordCount;
}

int main() {
    char str[100];

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

    int wordCount = countWords(str);

    // Print the word count
    printf("Number of words: %d\n", wordCount);

    return 0;
}
```

23. 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>

// Function to remove duplicate characters
void removeDuplicates(char *str) {
   int length = strlen(str);
   int index = 0;

   for (int i = 0; i < length; 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];

   // Input the string
   printf("Enter a string: ");
   scanf("%99s", str);
```

```
    removeDuplicates(str);

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

    return 0;
}
```

24.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 findFirstNonRepeatingCharacter(const char *str) {
    int charCount[200] = {0};

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

    // Find the first non-repeating character
    for (int i = 0; str[i] != '\0'; i++) {
        if (charCount[(unsigned char)str[i]] == 1) {
```

```
            return str[i];
        }
    }

    return '\0'; // Return null character if all are repeating
}

int main() {
    char str[100];



    printf("Enter a string: ");
    scanf("%99s", str);



    char result = findFirstNonRepeatingCharacter(str);



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

    return 0;
}
```

25.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 <string.h>

int isNumericString(const char *str) {

    if (*str == '-') {
        str++;
    }

    while (*str) {
        if (*str < '0' || *str > '9') {
            return 0;
        }
        str++;
    }
    return 1;
}

int stringToInteger(const char *str) {
    int result = 0;
    int isNegative = 0;


    if (*str == '-') {
        isNegative = 1;
        str++;
    }
```

```
    while (*str) {
        result = result * 10 + (*str - '0');
        str++;
    }

    if (isNegative) {
        result = -result;
    }

    return result;
}

int main() {
    char str[100];


    printf("Enter a numeric string: ");
    scanf("%99s", str);


    if (isNumericString(str)) {

        int value = stringToInteger(str);
        printf("Converted integer value: %d\n", value);
    } else {
        printf("Error: Invalid numeric string.\n");
    }

    return 0;
```

}

\

26.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>


// Function to sort characters in a string

void sortString(char *str) {

   int len = strlen(str);

   for (int i = 0; i < len - 1; i++) {

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

         if (str[i] > str[j]) {

            // Swap the 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) {
    // Check if the lengths of the strings are equal
    if (strlen(str1) != strlen(str2)) {
        return 0;
    }

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

    // Compare sorted strings
    if (strcmp(str1, str2) == 0) {
        return 1;
    } else {
        return 0;
    }
}

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

    // Input the first string
    printf("Enter the first string: ");
    scanf("%99s", str1);

    // Input the second string
    printf("Enter the second string: ");
    scanf("%99s", str2);
```

```
    // 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;
}
```

27.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(const char *str1, const char *str2, char *mergedString) {
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    int i, j = 0;

    for (i = 0; i < len1 && i < len2; i++) {
        mergedString[j++] = str1[i];
        mergedString[j++] = str2[i];
    }
```

```
    while (i < len1) {
        mergedString[j++] = str1[i++];
    }

    while (i < len2) {
        mergedString[j++] = str2[i++];
    }

    mergedString[j] = '\0';
}

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

    printf("Enter the first string: ");
    scanf("%99s", str1);

    printf("Enter the second string: ");
    scanf("%99s", str2);

    // Merge the strings alternately
    mergeStringsAlternately(str1, str2, mergedString);
```

```
    // Print the merged string
    printf("Merged string: %s\n", mergedString);

    return 0;
}
```

28. 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>

int isVowel(char ch) {
    ch = (ch >= 'A' && ch <= 'Z') ? ch + ('a' - 'A') : ch; // Convert to lowercase
    return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');
}

int isAlpha(char ch) {
    return (ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z');
}

// Function to count consonants in a string
int countConsonants(const char *str) {
```

```
    int count = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (isAlpha(str[i]) && !isVowel(str[i]) && str[i] != ' ') {
            count++;
        }
    }
    return count;
}

int main() {
    char str[100];

    // Input the string
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    // Remove the newline character if it exists
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n') {
        str[len - 1] = '\0';
    }

    // Count the consonants in the string
    int consonantCount = countConsonants(str);

    // Print the count of consonants
    printf("Number of consonants: %d\n", consonantCount);

    return 0;
}
```

29.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 <stdlib.h>

#include <string.h>


// Function to replace all occurrences of target with replacement in main_str

void replace_substring(char *main_str, const char *target, const char *replacement) {

    char *result;

    char *ins;    // Pointer for the next insert point

    char *tmp;

    int len_target;

    int len_replacement;

    int len_front;

    int count;


    if (!main_str || !target) {

        return;

    }

    len_replacement = strlen(replacement);

    if (len_replacement == 0) {

        return;

    }

    len_target = strlen(target);

    if (len_target == 0) {

        return;

    }
```

```
    // Count the number of replacements needed
    ins = main_str;
    for (count = 0; (tmp = strstr(ins, target)); ++count) {
        ins = tmp + len_target;
    }

    // Allocate memory for the result string
    tmp = result = malloc(strlen(main_str) + (len_replacement - len_target) * count + 1);

    if (!result) {
        return;
    }

    // Perform the replacements
    while (count--) {
        ins = strstr(main_str, target);
        len_front = ins - main_str;
        tmp = strncpy(tmp, main_str, len_front) + len_front;
        tmp = strcpy(tmp, replacement) + len_replacement;
        main_str += len_front + len_target;
    }
    strcpy(tmp, main_str);

    // Copy the result back to main_str
    strcpy(main_str, result);
    free(result);
}

int main() {
    char main_str[1000];
    char target[100];
```

```
    char replacement[100];

    // Get input from user
    printf("Enter the main text: ");
    scanf(" %[^\n]%*c", main_str);  // Read main string with spaces

    printf("Enter the target substring: ");
    scanf(" %[^\n]%*c", target);  // Read target substring

    printf("Enter the replacement substring: ");
    scanf(" %[^\n]%*c", replacement);  // Read replacement substring

    // Replace all occurrences of target with replacement in main_str
    replace_substring(main_str, target, replacement);

    // Output the modified main text
    printf("Modified main text: %s\n", main_str);

    return 0;
}
```

30. 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_occurances(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);

    }
    return count;
}

int main(){
    const char *main_text = "This is important. Do this";
    const char *target_substring="test";
    int occurances = count_occurances(main_text,target_substring);
    printf("substring %s appears %d times \n",target_substring,occurances);
    return 0;

}
```

.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>

size_t custom_strlen(const char *str) {
```

```
    const char *ptr = str;

    while (*ptr != '\0') {
        ptr++;
    }

    return ptr - str;
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%99s", str);

    size_t length = custom_strlen(str);

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

    return 0;
}
```