1) Statistical Analysis Tool

a. Function Prototype: void computeStats(const double *array, int size, double *average, double *variance)

b. Data Types: const double*, int, double*

c. Concepts: Pointers, arrays, functions, passing constant data, pass by reference.

d. 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>
void computeStats(const double *array, int size, double *average, double *variance);
int main(){
   int size = 5;
   double array[] ={50.0,60.23,2.36,30.0,89};
   double average,variance;
   computeStats(array,size,&average,&variance);
   printf("The average is %.2f\n", average);
   printf("The variance is %.2f ",variance);
   return 0;
}
void computeStats(const double *array, int size, double *average, double *variance){
   double sum =0;
   double diff =0;
   for(int i=0;i<size;i++){  // average calulation
        sum += array[i];
   }
   *average = sum/size;
   for(int i=0;i<size;i++){  //varience
       diff += (array[i] -*average) *  (array[i] -*average);
   }
   *variance = diff /size;

}
```

2) Data Normalization

a. Function Prototype: double* normalizeData(const double *array, int size)

b. Data Types: const double*, int, double*

c. Concepts: Arrays, functions returning pointers, loops.

d. 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 = 5;
    double array[]={56.2,48.3,88.0,70.6,96.2};
    double *arr = normalizeData(array,size);
    printf("Normalized Data:");
    for(int i=0;i<size;i++){
        printf("%.2f \t",arr[i]);
    }
    return 0;
}
double* normalizeData(const double *array, int size){
    double min =array[0];
    double max = array[0];
    for(int i=0;i<size;i++){
        if(array[i]>max){
            max = array[i];
        }
        if(array[i]<min){
            min = array[i];
        }
    }
    double *arr = (double *)malloc(size * sizeof(double));
```

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

3) Experimental Report Generator

a. Function Prototype: void generateReport(const double *results, const char *descriptions[], int size)

b. Data Types: const double*, const char*[], int

c. Concepts: Strings, arrays, functions, passing constant data.

d. 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);
int main(){
    int size =5;
    double results[] = {97.5,88.3,66.5,77.3,55.5};
    const char *description[] ={
        "Experiment 1: Ampere circuital Law",
        "Experiment 2: Darwin's Theory",
        "Experiment 3: Chemical reaction efficiency",
        "Experiment 4: Thermal Conductivity",
        "Experiment 5: Resistence"
    };
    generateReport(results,description,size);
    return 0;
}
void generateReport(const double *results, const char *descriptions[], int size){
    printf("Experimental Reports : \n");
```

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

4) Data Anomaly Detector

a. Function Prototype: void detectAnomalies(const double *data, int size, double threshold, int *anomalyCount)

b. Data Types: const double*, int, double, int*

c. Concepts: Decision-making, arrays, pointers, functions.

d. 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);
int main(){
    int size = 5;
    double data[] ={12.5, 25.0, 7.3, 35.5, 50.1};
    double threshold = 25;
    int anomalyCount =0;
    detectAnomalies(data,size,threshold,&anomalyCount);
    printf("Number of anomalies detected : %d\n",anomalyCount);
}
void detectAnomalies(const double *data, int size, double threshold, int *anomalyCount){
    *anomalyCount =0;
    printf("Anomaly Detection Report:\n");
    printf("==============================\n");
    for(int i=0;i<size;i++){
        if(data[i]<threshold){
            (*anomalyCount)++;
            printf("Anomaly detected at index %d : %.2f exceeds the thershold%.2f
\n",i,data[i],threshold);
```

```
        }
        else{
            printf("Anomaly Not detected \n");
        }
    }

}
```

5) Data Classifier

a. Function Prototype: void classifyData(const double *data, int size, char *labels[], double threshold)

b. Data Types: const double*, int, char*[], double

c. Concepts: Decision-making, arrays, functions, pointers.

d. 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 =5;
    const double data []= {12.5, 25.0, 7.3, 35.5, 50.1};
    double threshold = 25.00;
    char *labels[size];
    classifyData(data,size,labels,threshold);
    printf("Data classification Report :\n");
    printf("==========================\n");
    for(int i=0;i<size;i++){
        printf("Data points %.2f : %s\n",data[i],labels[i]);
    }
    return 0;
}
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

a. Function Prototype: void adjustWeights(double *weights, int size, double learningRate)

b. Data Types: double*, int, double

c. Concepts: Pointers, arrays, functions, loops.

d. 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);
int main(){
    int size =5;
    double weights[] = {3.2,45.9,-0.5,2.6,-0.7};
    double learningRate =0.1;
    printf("Original weights \n");
    for (int i = 0; i < size; i++) {
        printf("%.2f ", weights[i]);
    }
    printf("\n");
    adjustWeights(weights,size,learningRate);
```

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

7) AI Model Evaluator

a. Function Prototype: void evaluateModels(const double *accuracies, int size, double *bestAccuracy)

b. Data Types: const double*, int, double*

c. Concepts: Loops, arrays, functions, pointers.

d. 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);
int main(){
    int size =5;
    const double accuracies[] = {97.5,88.3,66.5,77.3,55.5};
    double bestAccuracy = 0.0;
    evaluateModels(accuracies,size,&bestAccuracy);
    printf("The best accurary is : %.2f%%\n ",bestAccuracy);
    return 0;
}
void evaluateModels(const double *accuracies, int size, double *bestAccuracy){
```

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

}
```

8) Decision Tree Constructor

a. Function Prototype: void constructDecisionTree(const double *features, int size, int *treeStructure)

b. Data Types: const double*, int, int*

c. Concepts: Decision-making, arrays, functions.

d. 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);

int main(){
    int size = 5;
    const double features[] ={25.5,88.3,45.5,77.3,55.5};
    int treeStructure[5] ={0};
    constructDecisionTree(features,size,treeStructure);
    printf("Decision tree Structure \n");
    printf("============================\n");
    for(int i=0;i<size;i++){
        printf("Node %d : %d \n",i,treeStructure[i]);
    }
}
```

```
void constructDecisionTree(const double *features, int size, int *treeStructure){
    double threshold =50.0;
    for(int i=0;i<size;i++){
        if(features[i]>threshold){
            treeStructure[i] =1; //right
        }
        else{
            treeStructure[i] =0;  //left
        }
    }

}
```

9) Sentiment Analysis Processor

a. Function Prototype: void processSentiments(const char *sentences[], int size, int *sentimentScores)

b. Data Types: const char*[], int, int*

c. Concepts: Strings, arrays, functions, pointers.

d. 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);
int main(){
    int size =5;
    const char *sentences[] ={
        "I love Ice cream !",
        "Yesterday it was bad day",
        "Mr witch is very bad person",
        "I future I will buy a fantastic sports car",
```

```
        "New Year party was good but it has become worst experience!"
    };
    int sentimentScores[5] ={0};
    processSentiments(sentences,size,sentimentScores);
    printf("Analysed Sentiment Scores \n");
    printf("================================\n");
    for(int i=0;i<size;i++){
        printf("Sentence %d : %d \n",i+1,sentimentScores[i]);
    }
    return 0;
}
void processSentiments(const char *sentences[], int size, int *sentimentScores){
    int score =0;
    for(int i=0;i<size;i++){
       if (strstr(sentences[i], "love") || strstr(sentences[i], "fantastic") || strstr(sentences[i], "great") ||
strstr(sentences[i], "excellent")) {
            score += 1;
        }

        if (strstr(sentences[i], "worst") || strstr(sentences[i], "bad") || strstr(sentences[i], "terrible") ||
strstr(sentences[i], "poor")) {
            score -= 1;
        }
        sentimentScores[i]=score;
    }

}
```

10) Training Data Generator

a. Function Prototype: double* generateTrainingData(const double *baseData, int size, int multiplier)

b. Data Types: const double*, int, double*

c. Concepts: Arrays, functions returning pointers, loops.

d. 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);

int main() {
    int size = 5;
    double baseData[] = {1.2, 2.5, 3.7, 4.8, 5.9};
    int multiplier = 3;
    double *trainingData = generateTrainingData(baseData, size, multiplier);

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

    return 0;
}

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

```
    double *trainingData = (double *)malloc(size * sizeof(double));// Allocate memory for the new training data array
    if (trainingData == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    for (int i = 0; i < size; i++) {            // Generate training data by applying the multiplier
        trainingData[i] = baseData[i] * multiplier;
    }
    return trainingData;
}
```

Computer Vision

11) Image Filter Application

a. Function Prototype: void applyFilter(const unsigned char *image, unsigned char *filteredImage, int width, int height)

b. Data Types: const unsigned char*, unsigned char*, int

c. Concepts: Arrays, pointers, functions.

d. Details: Apply a filter to an image, modifying the filtered image by reference.

```
#include<stdio.h>
void applyFilter(const unsigned char *image, unsigned char *filteredImage, int width, int height);
int main(){
    int width =3,height =3;
    const unsigned char image[] ={
        100, 120, 130,
        150, 180, 200,
        220, 240, 250
    };
    unsigned char filteredImage[9];
    applyFilter(image,filteredImage,width,height);
```

```
    printf("Filtered Image \n");
    for(int i=0;i<height;i++){
        for(int j=0;j<width;j++){
            printf("%3d ",filteredImage[i*width+j]);
        }
        printf("\n");
    }
  return 0;
}
void applyFilter(const unsigned char *image, unsigned char *filteredImage, int width, int height){
    for (int row = 0; row < height; row++) {
        for (int col = 0; col < width; col++) {
            int sum = 0, count = 0;

            // Apply a simple average filter (considering neighbors)
            for (int i = -1; i <= 1; i++) {
                for (int j = -1; j <= 1; j++) {
                    int neighborRow = row + i;
                    int neighborCol = col + j;

                    // Check if the neighbor is within bounds
                    if (neighborRow >= 0 && neighborRow < height && neighborCol >= 0 &&
neighborCol < width) {
                        sum += image[neighborRow * width + neighborCol];
                        count++;
                    }
                }
            }

            // Calculate the average and assign it to the filtered image
            filteredImage[row * width + col] = sum / count;
```

```
        }
    }
}
```

12) Edge Detection Algorithm

a. Function Prototype: void detectEdges(const unsigned char *image, unsigned char *edges, int width, int height)

b. Data Types: const unsigned char*, unsigned char*, int

c. Concepts: Loops, arrays, decision-making, functions.

d. Details: Detect edges in an image, updating the edges array by reference.

```
#include <stdio.h>
#include <stdlib.h>
#include<math.h>
void detectEdges(const unsigned char *image, unsigned char *edges, int width, int height);

int main() {
    int width = 3, height = 3;
    unsigned char image[] = {100, 120, 130, 150, 180, 200, 220, 240, 250};
    unsigned char edges[9];

    detectEdges(image, edges, width, height);

    printf("Edge Detected Image:\n");
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            printf("%3d ", edges[i * width + j]);
        }
        printf("\n");
    }
```

```
    return 0;
}

void detectEdges(const unsigned char *image, unsigned char *edges, int width, int height) {
    int Gx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
    int Gy[3][3] = {{-1, -2, -1}, { 0,  0,  0}, { 1,  2,  1}};

    for (int row = 1; row < height - 1; row++) {
        for (int col = 1; col < width - 1; col++) {
            int gradientX = 0, gradientY = 0;
            for (int i = -1; i <= 1; i++) {
                for (int j = -1; j <= 1; j++) {
                    gradientX += image[(row + i) * width + (col + j)] * Gx[i + 1][j + 1];
                    gradientY += image[(row + i) * width + (col + j)] * Gy[i + 1][j + 1];
                }
            }
            int magnitude = (int)sqrt(gradientX * gradientX + gradientY * gradientY);
            if (magnitude > 255) magnitude = 255;
            edges[row * width + col] = (unsigned char)magnitude;
        }
    }

    for (int i = 0; i < width; i++) {
        edges[i] = 0;
        edges[(height - 1) * width + i] = 0;
    }
    for (int i = 0; i < height; i++) {
        edges[i * width] = 0;
        edges[i * width + (width - 1)] = 0;
    }
```

```
}
```

13) Object Recognition System

a. Function Prototype: void recognizeObjects(const double *features, int size, char *objectLabels[])

b. Data Types: const double*, int, char*[]

c. Concepts: Decision-making, arrays, functions, pointers.

d. 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[]);
int main(){
    int size =5;
    const double features[] = {1.5,2.3,0.7,0.65,-0.75};
    char *objectLabels[] = {"", "", "", "", ""};
    recognizeObjects(features,size,objectLabels);
    printf("Object labels \n");
    for(int i=0;i<size;i++){
        printf("Features %d : %s \n",i+1,objectLabels[i]);
    }
    return 0;
}
void recognizeObjects(const double *features, int size, char *objectLabels[]){
    for(int i=0;i<size;i++){
        if(features[i]>0.9){
            objectLabels[i] = "Birds";
        }
        else if (features[i]<0.9 && features[i]>0){
```

```
            objectLabels[i] ="Animals";


        }else if(features[i]<0){
            objectLabels[i] = "Unknown";
        }
    }
}
```

14) Image Resizing Function

a. Function Prototype: void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth, int originalHeight, int newWidth, int newHeight)

b. Data Types: const unsigned char*, unsigned char*, int

c. Concepts: Arrays, functions, pointers.

d. Details: Resize an image to new dimensions, modifying the output image by reference.

```
#include <stdio.h>
#include <string.h>
void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth, int originalHeight, int newWidth, int newHeight);


int main(){
    int originalWidth =4,originalHeight =4;
    int newWidth =2,newHeight =2;
    const unsigned char inputImage[9] ={
        255,0,255,
        0,255,0,
        255,0,255
    };
    unsigned char outputImage[4];
    resizeImage(inputImage,outputImage,originalWidth,originalHeight,newWidth,newHeight);
    printf("Resized image \n");
```

```
    printf("============================\n");
    for(int i=0;i<newHeight;i++){
        for(int j=0;j<newWidth;j++){
            printf("%d ",outputImage[i*newWidth+j]);
        }
        printf("\n");
    }
    return 0;
}
void resizeImage(const unsigned char *inputImage, unsigned char *outputImage, int originalWidth,
int originalHeight, int newWidth, int newHeight){
     float x = (float) originalWidth/newWidth;
     float y = (float) originalHeight/newHeight;
      for (int i = 0; i < newHeight; i++) {
           for (int j = 0; j < newWidth; j++) {
              int x = (int)(j * x);
              int y = (int)(i * y);
              outputImage[i * newWidth + j] = inputImage[y * originalWidth + x];
           }
    }

}
```

15) Color Balance Adjuster

a. Function Prototype: void balanceColors(const unsigned char *image, unsigned char *balancedImage, int width, int height)

b. Data Types: const unsigned char*, unsigned char*, int

c. Concepts: Arrays, functions, pointers, loops.

d. 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);

int main() {
    int width = 3, height = 3; // Image dimensions

    unsigned char image[27] = { // Original image (3x3 pixels)
        255, 100, 100, 200, 150, 50, 100, 200, 255,
        150, 150, 150, 0, 0, 255, 0, 255, 0,
        255, 255, 0, 255, 0, 255, 0, 0, 0
    };
    unsigned char balancedImage[27]; // To store the balanced image
    balanceColors(image, balancedImage, width, height); // Apply color balance
    printf("Balanced Image:\n");
    for (int i = 0; i < height; i++) { // Iterate over each row
        for (int j = 0; j < width; j++) { // Iterate over each column
            printf("(%d, %d, %d) ",
                balancedImage[(i * width + j) * 3],
                balancedImage[(i * width + j) * 3 + 1],
                balancedImage[(i * width + j) * 3 + 2]); // Print RGB values
        }
        printf("\n");
    }

    return 0;
}

void balanceColors(const unsigned char *image, unsigned char *balancedImage, int width, int height)
{
    float redBalance = 1.2, greenBalance = 0.9, blueBalance = 1.1; // Balance factors for each color channel

    for (int i = 0; i < height; i++) { // Iterate over each row
```

```
        for (int j = 0; j < width; j++) { // Iterate over each column
            int index = (i * width + j) * 3; // Pixel index

            balancedImage[index] = (unsigned char)(image[index] * redBalance); // Adjust red channel
            balancedImage[index + 1] = (unsigned char)(image[index + 1] * greenBalance); // Adjust green channel
            balancedImage[index + 2] = (unsigned char)(image[index + 2] * blueBalance); // Adjust blue channel

            // Clamp values to ensure they stay within the valid range [0, 255]
            if (balancedImage[index] > 255) balancedImage[index] = 255;
            if (balancedImage[index + 1] > 255) balancedImage[index + 1] = 255;
            if (balancedImage[index + 2] > 255) balancedImage[index + 2] = 255;
        }
    }
}
```

16) Pattern Recognition Algorithm

a. Function Prototype: void recognizePatterns(const char *patterns[], int size, int *matchCounts)

b. Data Types: const char*[], int, int*

c. Concepts: Strings, arrays, decision-making, pointers.

d. 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++) {
        int count = 0;
        for (int j = 0; j < size; j++) {
```

```
            if (strcmp(patterns[i], patterns[j]) == 0) {
                count++;
            }
        }
        matchCounts[i] = count;
    }
}

int main() {
    const char *patterns[] = {"apple", "banana", "apple", "orange"};
    int size = 4;
    int matchCounts[size];

    recognizePatterns(patterns, size, matchCounts);

    for (int i = 0; i < size; i++) {
        printf("Pattern: %s, Count: %d\n", patterns[i], matchCounts[i]);
    }

    return 0;
}
```

17) Climate Data Analyzer

a. Function Prototype: void analyzeClimateData(const double *temperatureReadings, int size, double *minTemp, double *maxTemp)

b. Data Types: const double*, int, double*

c. Concepts: Decision-making, arrays, functions.

d. 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 = 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() {
    double temperatures[] = {23.5, 26.8, 20.4, 29.7, 25.3};
    int size = 5;
    double minTemp, maxTemp;

    analyzeClimateData(temperatures, size, &minTemp, &maxTemp);

    printf("Min Temperature: %.2f, Max Temperature: %.2f\n", minTemp, maxTemp);

    return 0;
}
```

18) Quantum Data Processor

a. Function Prototype: void processQuantumData(const double *measurements, int size, double *processedData)

b. Data Types: const double*, int, double*

c. Concepts: Arrays, functions, pointers, loops.

d. 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] * 1.1; // Example processing: scale by 1.1
    }
}

int main() {
    double measurements[] = {1.2, 3.4, 2.5};
    int size = 3;
    double processedData[size];

    processQuantumData(measurements, size, processedData);

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

    return 0;
}
```

19) Scientific Data Visualization

a. Function Prototype: void visualizeData(const double *data, int size, const char *title)

b. Data Types: const double*, int, const char*

c. Concepts: Arrays, functions, strings.

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

int main() {
    double data[] = {1.1, 2.2, 3.3, 4.4};
    int size = 4;
    const char *title = "Scientific Data Visualization";

    visualizeData(data, size, title);

    return 0;
}
```

20) Genetic Data Simulator

a. Function Prototype: double* simulateGeneticData(const double *initialData, int size, double mutationRate)

b. Data Types: const double*, int, double

c. Concepts: Arrays, functions returning pointers, loops.

d. 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;  // Simulating mutation
    }
    return simulatedData;
}

int main() {
    double initialData[] = {0.5, 1.2, 0.8};
    int size = 3;
    double mutationRate = 1.1;

    double *simulatedData = simulateGeneticData(initialData, size, mutationRate);

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

    free(simulatedData);

    return 0;
}
```

21) AI Performance Tracker

a. Function Prototype: void trackPerformance(const double *performanceData, int size, double *maxPerformance, double *minPerformance)

b. Data Types: const double*, int, double*

c. Concepts: Arrays, functions, pointers.

d. 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 = 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() {
    double performance[] = {98.5, 87.2, 91.3, 96.7};
    int size = 4;
    double maxPerformance, minPerformance;

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

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

    return 0;
```

```
}
```

22) Sensor Data Filter

a. Function Prototype: void filterSensorData(const double *sensorData, double *filteredData, int size, double filterThreshold)

b. Data Types: const double*, double*, int, double

c. Concepts: Arrays, functions, decision-making.

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

```
#include <stdio.h>

void filterSensorData(const double *sensorData, double *filteredData, int size, double filterThreshold) {
    int index = 0;
    for (int i = 0; i < size; i++) {
        if (sensorData[i] >= filterThreshold) {
            filteredData[index++] = sensorData[i];
        }
    }
}

int main() {
    double sensorData[] = {1.2, 3.4, 2.5, 0.8, 4.0};
    int size = 5;
    double filterThreshold = 2.0;
    double filteredData[size];

    filterSensorData(sensorData, filteredData, size, filterThreshold);

    for (int i = 0; i < size; i++) {
        if (filteredData[i] != 0) {
```

```
            printf("Filtered Data[%d]: %.2f\n", i, filteredData[i]);
        }
    }

    return 0;
}
```

23) Logistics Data Planner

a. Function Prototype: void planLogistics(const double *resourceLevels, double *logisticsPlan, int size)

b. Data Types: const double*, double*, int

c. Concepts: Arrays, functions, pointers, loops.

d. 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] * 0.8;  // Planning based on resource levels
    }
}

int main() {
    double resourceLevels[] = {100.0, 200.0, 300.0};
    int size = 3;
    double logisticsPlan[size];

    planLogistics(resourceLevels, logisticsPlan, size);

    for (int i = 0; i < size; i++) {
```

```
        printf("Logistics Plan[%d]: %.2f\n", i, logisticsPlan[i]);
    }

    return 0;
}
```

24) Satellite Image Processor

a. Function Prototype: void processSatelliteImage(const unsigned char *imageData, unsigned char *processedImage, int width, int height)

b. Data Types: const unsigned char*, unsigned char*, int

c. Concepts: Arrays, functions, pointers, loops.

d. 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 processing: reducing brightness
    }
}

int main() {
    unsigned char imageData[] = {255, 128, 64, 32, 16}; // Example pixel values
    int width = 5, height = 1; // Simplified example with a 1D array
    unsigned char processedImage[5];

    processSatelliteImage(imageData, processedImage, width, height);

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

```
    return 0;
}
```

25) Flight Path Analyzer

a. Function Prototype: void analyzeFlightPath(const double *pathCoordinates, double *optimizedPath, int size)

b. Data Types: const double*, double*, int

c. Concepts: Arrays, functions, pointers, loops.

d. 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 optimization: reducing each coordinate by 10%
    }
}

int main() {
    double pathCoordinates[] = {100.0, 200.0, 300.0};
    int size = 3;
    double optimizedPath[size];

    analyzeFlightPath(pathCoordinates, optimizedPath, size);

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

```
    return 0;
}
```

26) AI Data Augmenter

a. Function Prototype: void augmentData(const double *originalData, double *augmentedData, int size, double augmentationFactor)

b. Data Types: const double*, double*, int, double

c. Concepts: Arrays, functions, pointers, loops.

d. 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;  // Example: applying the augmentation factor
    }
}

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);

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

```
    return 0;
}
```

27) Medical Image Analyzer

a. Function Prototype: void analyzeMedicalImage(const unsigned char *imageData, unsigned char *analysisResults, int width, int height)

b. Data Types: const unsigned char*, unsigned char*, int

c. Concepts: Arrays, functions, pointers, loops.

d. 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: binarize the image
    }
}

int main() {
    unsigned char imageData[] = {255, 128, 64, 32, 16};
    int width = 5, height = 1;
    unsigned char analysisResults[5];

    analyzeMedicalImage(imageData, analysisResults, width, height);

    for (int i = 0; i < width * height; i++) {
        printf("Analysis Results[%d]: %u\n", i, analysisResults[i]);
    }
```

```
    return 0;
}
```

28) Object Tracking System

a. Function Prototype: void trackObjects(const double *objectData, double *trackingResults, int size)

b. Data Types: const double*, double*, int

c. Concepts: Arrays, functions, pointers, loops.

d. 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] * 1.05;  // Example: increasing object position by 5%
    }
}

int main() {
    double objectData[] = {1.2, 3.4, 5.6};
    int size = 3;
    double trackingResults[size];

    trackObjects(objectData, trackingResults, size);

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

    return 0;
```

```
}
```

29) Defense Strategy Optimizer

a. 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] * 0.75;  // Example: reducing threat levels by 25%
    }
}

int main() {
    double threatLevels[] = {50.0, 70.0, 30.0};
    int size = 3;
    double optimizedStrategies[size];

    optimizeDefenseStrategy(threatLevels, optimizedStrategies, size);

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

    return 0;
}
```