```
/*
Requirements
1. Define Data Types
Object Structure:
1...Use a structure DetectedObject to represent the detected objects with the
following fields:
objectID (integer): Unique identifier for the detected object.
label (string): Label assigned to the object (e.g., "Car", "Person", "Dog").
confidence (float): Confidence score of detection (e.g., 0.95 for 95%).
2...boundingBox (structure): Define a nested structure to represent the
bounding box of the detected object:
x (integer): Top-left x-coordinate of the box.
y (integer): Top-left y-coordinate of the box.
width (integer): Width of the bounding box.
height (integer): Height of the bounding box.
Union for Object Properties:
Use a union ObjectProperties to store additional object-related properties:
speed (float): Speed of the object (e.g., for vehicles or moving objects).
area (float): Area of the bounding box (calculated as width × height).

2. Features
Dynamic Memory Allocation:
Dynamically allocate memory for an array of DetectedObject structures based on
user input for the number of detected objects.
Input and Output:
Input object details, including their bounding box coordinates, label, and
confidence score.
Calculate and store either the speed or area using the ObjectProperties union
based on the object's type.
Display:
Display details of all detected objects, including their labels, bounding
boxes, and associated properties (speed or area).
Search:
Search for an object by its objectID and display its details.
Sorting:
Sort objects by their confidence scores in descending order.

3. Typedef
Use typedef to define aliases for DetectedObject, BoundingBox, and
ObjectProperties to simplify the code.
Program Requirements
1. Menu Options
Input Detected Object Details.
Display All Detected Objects.
Search Detected Object by ID.
Sort Objects by Confidence Score.(in discending order)
Calculate Object Properties (Speed or Area).
Exit.
```

```c
has context menu


*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    int x; // Top-left x-coordinate
    int y; // Top-left y-coordinate
    int width; // Width of the bounding box
    int height; // Height of the bounding box
} BoundingBox;

typedef union {
    float speed; // Speed of the object
    float area;  // Area of the bounding box
} ObjectProperties;

typedef struct {
    int objectID;          // Unique identifier
    char label[100];       // Label assigned to the object
    float confidence;      // Confidence score of detection
    BoundingBox bbox;      // Bounding box structure
    ObjectProperties props; // Additional properties
} DetectedObject;

void InputDetectedObject(DetectedObject *detected, int *count);
void displayObjects(DetectedObject *detected, int count);
void searchObjects(DetectedObject *detected, int count);
void sortObjectsByConfidence(DetectedObject *detected, int count);
void calculateObjectProperties(DetectedObject *detected, int count);

int main() {
    int n, count = 0;
    printf("\nEnter the number of detected objects: ");
    scanf("%d", &n);

    DetectedObject *detected = (DetectedObject *)malloc(n *
sizeof(DetectedObject));
    if (!detected) {
        printf("Memory allocation failed!");
        return 1;
    }

    int choice;
    do {
        printf("\nSelect from the menu:\n"
               "1. Input Detected Object Details\n"
               "2. Display All Detected Objects\n"
               "3. Search Detected Object by ID\n"
               "4. Sort Objects by Confidence Score (Descending)\n"
               "5. Calculate Object Properties (Speed or Area)\n"
               "6. Exit\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (count < n) {
                    InputDetectedObject(detected, &count);
                } else {
                    printf("\nCapacity to store objects reached!");
                }
                break;
            case 2:
                displayObjects(detected, count);
                break;
            case 3:
                searchObjects(detected, count);
                break;
            case 4:
                sortObjectsByConfidence(detected, count);
                break;
            case 5:
                calculateObjectProperties(detected, count);
                break;
            case 6:
                printf("\nExiting...\n");
                break;
            default:
                printf("\nInvalid choice! Please try again.");
        }
    } while (choice != 6);

    free(detected);
    return 0;
}

void InputDetectedObject(DetectedObject *detected, int *count) {
    printf("\nEnter Object ID: ");
    scanf("%d", &detected[*count].objectID);
    getchar(); // Consume newline
    printf("Enter Label: ");
    fgets(detected[*count].label, 100, stdin);
    detected[*count].label[strcspn(detected[*count].label, "\n")] = '\0'; //
Remove newline
```

```c
    printf("Enter Confidence Score (0 to 1): ");
    scanf("%f", &detected[*count].confidence);
    printf("Enter Bounding Box (x, y, width, height): ");
    scanf("%d %d %d %d", &detected[*count].bbox.x, &detected[*count].bbox.y,
          &detected[*count].bbox.width, &detected[*count].bbox.height);

    printf("Enter Speed (if applicable, 0 if not): ");
    scanf("%f", &detected[*count].props.speed);
    (*count)++;
}

void displayObjects(DetectedObject *detected, int count) {
    printf("\nDetected Objects:\n");
    for (int i = 0; i < count; i++) {
        printf("ID: %d, Label: %s, Confidence: %.2f, Bounding Box: (%d, %d,
%d, %d), Speed: %.2f\n",
               detected[i].objectID, detected[i].label,
detected[i].confidence,
               detected[i].bbox.x, detected[i].bbox.y,
               detected[i].bbox.width, detected[i].bbox.height,
               detected[i].props.speed);
    }
}

void searchObjects(DetectedObject *detected, int count) {
    int id;
    printf("\nEnter Object ID to search: ");
    scanf("%d", &id);

    for (int i = 0; i < count; i++) {
        if (detected[i].objectID == id) {
            printf("ID: %d, Label: %s, Confidence: %.2f, Bounding Box: (%d,
%d, %d, %d), Speed: %.2f\n",
                   detected[i].objectID, detected[i].label,
detected[i].confidence,
                   detected[i].bbox.x, detected[i].bbox.y,
                   detected[i].bbox.width, detected[i].bbox.height,
                   detected[i].props.speed);
            return;
        }
    }
    printf("\nObject with ID %d not found!", id);
}

void sortObjectsByConfidence(DetectedObject *detected, int count) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (detected[j].confidence < detected[j + 1].confidence) {
                DetectedObject temp = detected[j];
                detected[j] = detected[j + 1];
                detected[j + 1] = temp;
            }
        }
    }
    printf("\nObjects sorted by confidence successfully!");
}

void calculateObjectProperties(DetectedObject *detected, int count) {
    for (int i = 0; i < count; i++) {
        detected[i].props.area = detected[i].bbox.width *
detected[i].bbox.height;
        printf("ID: %d, Label: %s, Area: %.2f\n", detected[i].objectID,
detected[i].label, detected[i].props.area);
    }
}
```