ASSESSMENT 2 ANSWER

Requirements

1. Define Data Types

1. **Location Structure:**
   Define a structure Location to store details about a geographic location, including:
   - **locationID (integer): Unique identifier for the location.**
   - **name (string): Name of the location (e.g., "Mount Everest", "Nile River").**
   - **latitude (float): Latitude coordinate.**
   - **longitude (float): Longitude coordinate.**
   - **altitude (float): Altitude in meters above sea level.**
2. **Union for Feature Details:**
   Use a union FeatureDetails to store additional information about the location based on its type:
   - **population (integer): For cities or populated areas.**
   - **area (float): For physical features like rivers, lakes, or mountains.**
3. **Feature Type Enumeration:**
   Use an enumeration FeatureType to classify locations:
   - **CITY, MOUNTAIN, RIVER, or LAKE.**

2. Features

- **Dynamic Memory Allocation:**
  Dynamically allocate memory for an array of Location structures based on the number of geographic features.
- **Input and Output:**
  - **Input the details of each location, including its type, coordinates, and relevant feature details.**
  - **Display the details of all locations, categorized by their types.**
- **Search:**
  - **Search for a location by its name or ID and display its details.**
- **Sorting:**
  - **Sort locations based on their altitude in descending order.**
- **Geospatial Analysis:**
  - **Calculate the distance between two geographic locations based on their latitude and longitude (Haversine formula).**

3. Typedef

Use typedef to define aliases for the Location and FeatureDetails structures and the FeatureType enumeration.

1. Menu Options

1. **Input Geographic Feature Details.**
2. **Display All Features Categorized by Type.**
3. **Search for a Feature by Name or ID.**
4. **Sort Features by Altitude.**
5. **Calculate Distance Between Two Features.**
6. **Exit.**

ANSWER:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<math.h>
#define R 6371.0

typedef struct {
    int locationID;
    char name[50];
    float latitude;
    float longitude;
    float altitude;

    union {
        int population;
        float area;
    } FeatureDetails;

    enum {
        CITY,
        MOUNTAIN,
        RIVER,
        LAKE
    } type;
} location;

void inputDetails(location *locations, int count);
void displayFeatures(location *locations, int count);
void searchFeature(location *locations, int count);
```

```
void sortFeatures(location *locations, int count);
void calculateDistanceMenu(location *locations, int count);
float calculateDistance(float lat1, float lon1, float lat2, float lon2);

int main()
{
    location *locations = NULL;
    int choice, count = 0;

    while (1)
    {
        printf("\nChoose from menu:\n");
        printf("1. Input Geographic Feature Details\n");
        printf("2. Display All Features Categorized by Type\n");
        printf("3. Search for a Feature by Name or ID\n");
        printf("4. Sort Features by Altitude\n");
        printf("5. Calculate Distance Between Two Features\n");
        printf("6. Exit\n");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter number of features: ");
                scanf("%d", &count);
                locations = (location *)malloc(count * sizeof(location));
                inputDetails(locations, count);
                break;
            case 2:
                displayFeatures(locations, count);
                break;
            case 3:
                searchFeature(locations, count);
                break;
            case 4:
                sortFeatures(locations, count);
                break;
            case 5:
                calculateDistanceMenu(locations, count);
                break;
            case 6:
                free(locations);
                exit(0);
                break;
```

```
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}

void inputDetails(location *locations, int count)
{
    for (int i = 0; i < count; i++)
    {
        printf("\nEnter details for feature %d:\n", i + 1);
        printf("ID: ");
        scanf("%d", &locations[i].locationID);
        printf("Name: ");
        scanf("%s", locations[i].name);
        printf("Latitude: ");
        scanf("%f", &locations[i].latitude);
        printf("Longitude: ");
        scanf("%f", &locations[i].longitude);
        printf("Altitude: ");
        scanf("%f", &locations[i].altitude);
        printf("Type (0-CITY, 1-MOUNTAIN, 2-RIVER, 3-LAKE): ");
        scanf("%d",&locations[i].type);

        if (locations[i].type == 0)
        {
            printf("Population: ");
            scanf("%d", &locations[i].FeatureDetails.population);
        } else
        {
            printf("Area: ");
            scanf("%f", &locations[i].FeatureDetails.area);
        }
    }
}

void displayFeatures(location *locations, int count)
{
    const char *types[] = {"City", "Mountain", "River", "Lake"};
    for (int i = 0; i < 4; i++)
    {
        printf("\n%s:\n", types[i]);
        for (int j = 0; j < count; j++)
```

```
        {
            if (locations[j].type == i)
            {
                printf("ID: %d, Name: %s, Latitude: %.2f, Longitude: %.2f, Altitude: %.2f\n",
                       locations[j].locationID, locations[j].name,
                       locations[j].latitude, locations[j].longitude, locations[j].altitude);
                if (i == 0)
                {
                    printf("Population: %d\n", locations[j].FeatureDetails.population);
                } else {
                    printf("Area: %.2f\n", locations[j].FeatureDetails.area);
                }
            }
        }
    }
}

void searchFeature(location *locations, int count)
{
    char query[50];
    printf("Enter Name or ID to search: ");
    scanf("%s", query);

    for (int i = 0; i < count; i++)
    {
        if (atoi(query) == locations[i].locationID || strcmp(query, locations[i].name) == 0)
        {
            printf("Found: ID: %d, Name: %s, Latitude: %.2f, Longitude: %.2f, Altitude: %.2f\n",
                   locations[i].locationID, locations[i].name,
                   locations[i].latitude, locations[i].longitude, locations[i].altitude);
            return;
        }
    }
    printf("Feature not found.\n");
}

void sortFeatures(location *locations, int count)
{
    for (int i = 0; i < count - 1; i++)
    {
        for (int j = 0; j < count - i - 1; j++)
        {
            if (locations[j].altitude < locations[j + 1].altitude)
            {
```

```
            location temp = locations[j];
            locations[j] = locations[j + 1];
            locations[j + 1] = temp;
        }
      }
   }
   printf("Features sorted by altitude.\n");
}

float calculateDistance(float lat1, float lon1, float lat2, float lon2)
{
   lat1 = lat1 * (3.14 / 180.0);
   lon1 = lon1 * (3.14 / 180.0);
   lat2 = lat2 * (3.14 / 180.0);
   lon2 = lon2 * (3.14 / 180.0);

   float dLat = lat2 - lat1;
   float dLon = lon2 - lon1;

   float distance = dLat * dLat + dLon * dLon;
   return R * sqrt(distance);
}

void calculateDistanceMenu(location *locations, int count)
{
   int id1, id2;
   printf("Enter IDs of two features to calculate distance:\n");
   scanf("%d %d", &id1, &id2);

   location *loc1 = NULL, *loc2 = NULL;
   for (int i = 0; i < count; i++)
   {
      if (locations[i].locationID == id1) loc1 = &locations[i];
      if (locations[i].locationID == id2) loc2 = &locations[i];
   }

   if (loc1 && loc2)
   {
      float distance = calculateDistance(loc1->latitude, loc1->longitude,
                                         loc2->latitude, loc2->longitude);
      printf("Distance: %.2f km\n", distance);
   } else {
      printf("One or both features not found.\n");
   }
```

}
Outputs:

```
Choose from menu:
1. Input Geographic Feature Details
2. Display All Features Categorized by Type
3. Search for a Feature by Name or ID
4. Sort Features by Altitude
5. Calculate Distance Between Two Features
6. Exit
1
Enter number of features: 2

Enter details for feature 1:
ID: 2
Name: nanda
Latitude: 56
Longitude: 45
Altitude: 32
Type (0-CITY, 1-MOUNTAIN, 2-RIVER, 3-LAKE): 1
Area: 456

Enter details for feature 2:
ID: 12
Name: nan
Latitude: 4
Longitude: 6
Altitude: 7
Type (0-CITY, 1-MOUNTAIN, 2-RIVER, 3-LAKE): 0
Population: 45
```

```
Choose from menu:
1. Input Geographic Feature Details
2. Display All Features Categorized by Type
3. Search for a Feature by Name or ID
4. Sort Features by Altitude
5. Calculate Distance Between Two Features
6. Exit
2

City:
ID: 12, Name: nan, Latitude: 4.00, Longitude: 6.00, Altitude: 7.00
Population: 45

Mountain:
ID: 2, Name: nanda, Latitude: 56.00, Longitude: 45.00, Altitude: 32.00
Area: 456.00
```

```
Choose from menu:
1. Input Geographic Feature Details
2. Display All Features Categorized by Type
3. Search for a Feature by Name or ID
4. Sort Features by Altitude
5. Calculate Distance Between Two Features
6. Exit
3
Enter Name or ID to search: 2
Found: ID: 2, Name: nanda, Latitude: 56.00, Longitude: 45.00, Altitude: 32.00
```

```
Choose from menu:
1. Input Geographic Feature Details
2. Display All Features Categorized by Type
3. Search for a Feature by Name or ID
4. Sort Features by Altitude
5. Calculate Distance Between Two Features
6. Exit
5
Enter IDs of two features to calculate distance:
2 12
Distance: 7224.01 km
```