Problem Statement:

```
Define Data Types
Device Structure:
Use a structure IoTDevice to represent IoT devices with the following fields:
deviceID (integer): Unique identifier for the device.
deviceName (string): Name of the IoT device (e.g., "Smart Light").
status (char): Device status ('A' for Active, 'I' for Inactive).
batteryLevel (float): Current battery level percentage.
totalUptime (float): Total uptime in hours.

Network Configuration Structure:
Use a structure NetworkConfig to represent the network configuration for each
device:
SSID (string): WiFi network name.
IPaddress (string): Device IP address.
port (integer): Communication port.

Union for Device Communication Protocol:
Use a union CommunicationProtocol to store either:
protocolName (string): Name of the communication protocol (e.g., "MQTT").
frequency (float): Operating frequency in GHz (used in RF-based protocols).

2. Features
Dynamic Memory Allocation:
Dynamically allocate memory for:
An array of IoTDevice structures to manage the inventory of devices.
An array of NetworkConfig structures to store network configurations.
Input and Output:
Input device details (ID, name, status, battery level, uptime).
Input network configurations (SSID, IP address, port).
Input communication protocol details using the union CommunicationProtocol.
Display:
Display all IoT devices and their statuses, network configurations, and
communication protocols.
Search:
Search for a device by its ID and display its details.
Update:
Update the status, battery level, or network configuration of a device.
Sorting:
Sort devices based on their total uptime in descending order.

3. Typedef
```

```
Use typedef to define aliases for IoTDevice, NetworkConfig, and
CommunicationProtocol.

Program Requirements
1. Menu Options
Input IoT Device Details.
Input Network Configuration.
Update IoT Device Status or Battery Level.
Display All Device Details.
Search Device by ID.
Sort Devices by Uptime.
Exit
```

Program:

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

#define STATUS_HISTORY_SIZE 10 // Max number of status entries to store
#define LOW_BATTERY_THRESHOLD 20.0 // Low battery threshold
#define HIGH_BATTERY_THRESHOLD 80.0 // High battery threshold

typedef struct IoTDevice {
    int deviceID;
    char deviceName[25];
    char status; // 'A' for Active, 'I' for Inactive
    float batteryLevel;
    float totalUptime;
    int statusHistory[STATUS_HISTORY_SIZE]; // Array to store status history (1 =
Active, 0 = Inactive)
    int historyIndex; // Keeps track of the current position in the status
history
    struct NetworkConfig* config; // Pointer to the network config for this
device
} IoTDevice;

typedef struct NetworkConfig {
    char SSID[25];
    char IPaddress[25];
    int port;
} NetworkConfig;
```

```c
typedef union CommunicationProtocol {
    char protocolName[25];
    float frequency;
} CommunicationProtocol;

// Function prototypes
void inputDeviceDetails(IoTDevice*, int);
void inputConfigDetails(NetworkConfig*, int);
void updateDeviceStatus(IoTDevice*, int);
void displayDeviceDetails(IoTDevice*, int);
void searchDeviceByID(IoTDevice*, int);
void displayStatusHistory(IoTDevice*, int);
void checkBatteryConsumption(IoTDevice*, int);
void sortDevicesByUptime(IoTDevice*, int);
void updateNetworkConfiguration(IoTDevice*, int);

// Function to handle memory cleanup
void freeResources(IoTDevice*, NetworkConfig*, CommunicationProtocol*);

int main() {
    int noOfDevices, noOfConfigs;

    // Input number of devices and configurations
    printf("Enter the number of devices: ");
    scanf("%d", &noOfDevices);
    printf("Enter the number of configurations: ");
    scanf("%d", &noOfConfigs);

    // Allocate memory for devices and configurations
    IoTDevice* devices = (IoTDevice*)malloc(noOfDevices * sizeof(IoTDevice));
    NetworkConfig* configs = (NetworkConfig*)malloc(noOfConfigs *
sizeof(NetworkConfig));

    if (!devices || !configs) {
        printf("Memory allocation failed. Exiting program.\n");
        freeResources(devices, configs, NULL);
        return -1;
    }

    // Allocate network configurations for each device
    for (int i = 0; i < noOfDevices; i++) {
        devices[i].config = (NetworkConfig*)malloc(sizeof(NetworkConfig));  //
Allocate memory for the network config
```

```
        if (!devices[i].config) {
            printf("Memory allocation for network config failed.\n");
            freeResources(devices, configs, NULL);
            return -1;
        }
    }

    while (1) {
        int choice;
        printf("\nMenu Options:\n");
        printf("1. Input IoT Device Details\n");
        printf("2. Input Network Configuration\n");
        printf("3. Update IoT Device Status or Battery Level\n");
        printf("4. Display All Device Details\n");
        printf("5. Search Device by ID\n");
        printf("6. Display Status History of Device\n");
        printf("7. Check Battery Consumption\n");
        printf("8. Sort Devices by Uptime\n");
        printf("9. Update Network Configuration\n");
        printf("10. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                inputDeviceDetails(devices, noOfDevices);
                break;
            case 2:
                inputConfigDetails(configs, noOfConfigs);
                break;
            case 3:
                updateDeviceStatus(devices, noOfDevices);
                break;
            case 4:
                displayDeviceDetails(devices, noOfDevices);
                break;
            case 5:
                searchDeviceByID(devices, noOfDevices);
                break;
            case 6:
                displayStatusHistory(devices, noOfDevices);
                break;
            case 7:
                checkBatteryConsumption(devices, noOfDevices);
```

```
                break;
            case 8:
                sortDevicesByUptime(devices, noOfDevices);
                break;
            case 9:
                updateNetworkConfiguration(devices, noOfDevices);
                break;
            case 10:
                freeResources(devices, configs, NULL);
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}

// Input device details and initialize status history
void inputDeviceDetails(IoTDevice* devices, int noOfD) {
    for (int i = 0; i < noOfD; i++) {
        printf("\nDevice %d:\n", i + 1);
        printf("Device ID: ");
        scanf("%d", &devices[i].deviceID);
        getchar(); // To consume the newline character
        printf("Device name: ");
        scanf(" %[^\n]", devices[i].deviceName);
        printf("Status ('A' for Active, 'I' for Inactive): ");
        scanf(" %c", &devices[i].status);
        printf("Battery level (0-100): ");
        scanf("%f", &devices[i].batteryLevel);
        printf("Total uptime (in hours): ");
        scanf("%f", &devices[i].totalUptime);

        // Initialize status history to be empty
        devices[i].historyIndex = 0;
        devices[i].statusHistory[devices[i].historyIndex] = (devices[i].status ==
'A') ? 1 : 0; // Store 1 for Active, 0 for Inactive
    }
}

// Input network configuration details
void inputConfigDetails(NetworkConfig* config, int noOfC) {
    for (int i = 0; i < noOfC; i++) {
        printf("\nNetwork Config %d:\n", i + 1);
```

```
        printf("SSID: ");
        scanf(" %[^\n]", config[i].SSID);
        printf("IP address: ");
        scanf(" %[^\n]", config[i].IPaddress);
        printf("Port: ");
        scanf("%d", &config[i].port);
    }
}

// Update device status and store the status in history
void updateDeviceStatus(IoTDevice* device, int noOfD) {
    int id;
    printf("\nEnter device ID: ");
    scanf("%d", &id);

    for (int i = 0; i < noOfD; i++) {
        if (device[i].deviceID == id) {
            char newStatus;
            printf("Enter new status ('A' for Active, 'I' for Inactive): ");
            scanf(" %c", &newStatus);
            device[i].status = newStatus;

            // Store the status in the history
            if (device[i].historyIndex < STATUS_HISTORY_SIZE - 1) {
                device[i].historyIndex++;
            } else {
                // Shift history to make room for the new status
                for (int j = 0; j < STATUS_HISTORY_SIZE - 1; j++) {
                    device[i].statusHistory[j] = device[i].statusHistory[j + 1];
                }
            }
            // Add new status (1 for Active, 0 for Inactive)
            device[i].statusHistory[device[i].historyIndex] = (newStatus ==
'A') ? 1 : 0;
        }
    }
}

// Display all device details
void displayDeviceDetails(IoTDevice* device, int noOfD) {
    for (int i = 0; i < noOfD; i++) {
        printf("\nDevice %d details:\n", i + 1);
        printf("Device ID: %d\n", device[i].deviceID);
```

```
        printf("Device name: %s\n", device[i].deviceName);
        printf("Status ('A' for Active, 'I' for Inactive): %c\n",
device[i].status);
        printf("Battery level: %.2f\n", device[i].batteryLevel);
        printf("Total uptime: %.2f\n", device[i].totalUptime);
    }
}

// Search device by ID
void searchDeviceByID(IoTDevice* device, int noOfD) {
    int id;
    printf("\nEnter device ID to search: ");
    scanf("%d", &id);

    for (int i = 0; i < noOfD; i++) {
        if (device[i].deviceID == id) {
            printf("\nDevice found: \n");
            printf("Device ID: %d\n", device[i].deviceID);
            printf("Device name: %s\n", device[i].deviceName);
            printf("Status ('A' for Active, 'I' for Inactive): %c\n",
device[i].status);
            printf("Battery level: %.2f\n", device[i].batteryLevel);
            printf("Total uptime: %.2f\n", device[i].totalUptime);
        }
    }
}

// Display status history for a device
void displayStatusHistory(IoTDevice* device, int noOfD) {
    int id;
    printf("\nEnter device ID to view status history: ");
    scanf("%d", &id);

    for (int i = 0; i < noOfD; i++) {
        if (device[i].deviceID == id) {
            printf("\nStatus history for Device ID %d (%s):\n",
device[i].deviceID, device[i].deviceName);
            for (int j = 0; j <= device[i].historyIndex; j++) {
                printf("Time %d: %s\n", j + 1, device[i].statusHistory[j] == 1 ?
"Active" : "Inactive");
            }
        }
    }
```

```c
}

// Check battery consumption and alert if necessary
void checkBatteryConsumption(IoTDevice* device, int noOfD) {
    int id;
    printf("\nEnter device ID to check battery consumption: ");
    scanf("%d", &id);

    for (int i = 0; i < noOfD; i++) {
        if (device[i].deviceID == id) {
            printf("\nDevice Battery Level: %.2f%%\n", device[i].batteryLevel);

            // Check if battery level is too low or high
            if (device[i].batteryLevel < LOW_BATTERY_THRESHOLD) {
                printf("Alert: Battery level is low!\n");
            } else if (device[i].batteryLevel > HIGH_BATTERY_THRESHOLD) {
                printf("Alert: Battery level is high!\n");
            } else {
                printf("Battery level is normal.\n");
            }
        }
    }
}

// Sort devices by total uptime (descending)
void sortDevicesByUptime(IoTDevice* devices, int noOfD) {
    IoTDevice temp;
    for (int i = 0; i < noOfD - 1; i++) {
        for (int j = i + 1; j < noOfD; j++) {
            if (devices[i].totalUptime < devices[j].totalUptime) {
                temp = devices[i];
                devices[i] = devices[j];
                devices[j] = temp;
            }
        }
    }
}

// Update network configuration for a device
void updateNetworkConfiguration(IoTDevice* devices, int noOfD) {
    int id;
    printf("\nEnter device ID to update network configuration: ");
    scanf("%d", &id);
```

```
    for (int i = 0; i < noOfD; i++) {
        if (devices[i].deviceID == id) {
            printf("Enter new SSID: ");
            scanf(" %[^\n]", devices[i].config->SSID);
            printf("Enter new IP address: ");
            scanf(" %[^\n]", devices[i].config->IPaddress);
            printf("Enter new port: ");
            scanf("%d", &devices[i].config->port);
        }
    }
}

// Function to handle memory cleanup
void freeResources(IoTDevice* devices, NetworkConfig* configs,
CommunicationProtocol* comm) {
    for (int i = 0; i < 10; i++) { // assuming 10 devices
        free(devices[i].config); // Free each network config for the devices
    }
    free(devices);
    free(configs);
    free(comm);
}
```

Output:

PS C:\Users\betti\Desktop\Training\Day19_Assessment2> ./test6

Enter the number of devices: 2

Enter the number of configurations: 2

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 1

Device 1:

Device ID: 10

Device name: Fridge

Status ('A' for Active, 'I' for Inactive): A

Battery level (0-100): 45

Total uptime (in hours): 7

Device 2:

Device ID: 20

Device name: Wahing Machine

Status ('A' for Active, 'I' for Inactive): I

Battery level (0-100): 78

Total uptime (in hours): 3

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 2

Network Config 1:

SSID: ert

IP address: 678.huu.899

Port: 7000

Network Config 2:

SSID: fgh

IP address: 899.7890.899

Port: 3000

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 3

Enter device ID: 10

Enter new status ('A' for Active, 'I' for Inactive): I

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 4

Device 1 details:

Device ID: 10

Device name: Fridge

Status ('A' for Active, 'I' for Inactive): I

Battery level: 45.00

Total uptime: 7.00

Device 2 details:

Device ID: 20

Device name: Wahing Machine

Status ('A' for Active, 'I' for Inactive): I

Battery level: 78.00

Total uptime: 3.00

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 5

Enter device ID to search: 20

Device found:

Device ID: 20

Device name: Wahing Machine

Status ('A' for Active, 'I' for Inactive): I

Battery level: 78.00

Total uptime: 3.00

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 6

Enter device ID to view status history: 10

Status history for Device ID 10 (Fridge):

Time 1: Active

Time 2: Inactive

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 7

Enter device ID to check battery consumption: 20

Device Battery Level: 78.00%

Battery level is normal.

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 8

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 4

Device 1 details:

Device ID: 10

Device name: Fridge

Status ('A' for Active, 'I' for Inactive): I

Battery level: 45.00

Total uptime: 7.00

Device 2 details:

Device ID: 20

Device name: Wahing Machine

Status ('A' for Active, 'I' for Inactive): I

Battery level: 78.00

Total uptime: 3.00

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 9

Enter device ID to update network configuration: 23

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 9

Enter device ID to update network configuration: 20

Enter new SSID: frt

Enter new IP address: 345.789.990

Enter new port: 4000

Menu Options:

1. Input IoT Device Details

2. Input Network Configuration

3. Update IoT Device Status or Battery Level

4. Display All Device Details

5. Search Device by ID

6. Display Status History of Device

7. Check Battery Consumption

8. Sort Devices by Uptime

9. Update Network Configuration

10. Exit

Enter your choice: 10

PS C:\Users\betti\Desktop\Training\Day19_Assessment2>