```c
#include <stdio.h>
#include <stdlib.h>
// Define the state and action space
#define NUM_STATES 3
#define NUM_ACTIONS 2
// Linked list node to represent Q-values
typedef struct QNode {
    int state;
    int action;
    float qValue;
    struct QNode* next;
} QNode;

typedef struct QList {
    QNode* head;
} QList;
// Initialize Q-list
void initQList(QList* qList) {
    qList->head = NULL;
}
// Insert a new Q-node into the Q-list
void insertQNode(QList* qList, int state, int action, float qValue) {
    QNode* newNode = (QNode*)malloc(sizeof(QNode));
    newNode->state = state;
    newNode->action = action;
    newNode->qValue = qValue;
    newNode->next = qList->head;
    qList->head = newNode;
}

// Print Q-list
void printQList(QList* qList) {
    QNode* cur = qList->head;
    while (cur != NULL) {
        printf("State: %d, Action: %d, Q-Value: %.3f\n", cur->state, cur->action, cur->qVa
        cur = cur->next;
    }
}
// Q-learning algorithm
void qLearning(QList* qList, int cur_State, int action, float reward, float learningRate,
    // Find the Q-value for the current state-action pair
    QNode* cur = qList->head;
    while (cur != NULL) {
        if (cur->state == cur_State && cur->action == action) {
            // Update the Q-value using the Q-learning formula
            cur->qValue = (1 - learningRate) * cur->qValue + learningRate * (reward + disco
            return;
        }
        cur = cur->next;
    }

    // If the state-action pair doesn't exist in the Q-list, insert a new node
    insertQNode(qList, cur_State, action, reward);
}

    int main() {
    // Initialize Q-list
```

```
    QList qList;
    initQList(&qList);

    // Define smart home parameters
    int numEpisodes = 10;
    int cur_State, action;
    float reward, learningRate = 0.1, discountFactor = 0.9;

    // Run Q-learning for a certain number of episodes
    for (int episode = 0; episode < numEpisodes; ++episode) {
        // Simulate smart home environment (currentState, action, reward)
        cur_State = rand() % NUM_STATES;  // Replace with actual state calculation
        action = rand() % NUM_ACTIONS;        // Replace with actual action selection
        reward = rand() % 10;                 // Replace with actual reward calculation

        // Update Q-values using Q-learning algorithm
        qLearning(&qList, cur_State, action, reward, learningRate, discountFactor);

        // Print Q-list after each episode
        printf("Episode %d:\n", episode + 1);
        printQList(&qList);
        printf("--------------------------------\n");
    }

    return 0;
}
```