Problem Solving (DS19)

هذا البرنامج التدريبي مُصاغ بعناية لتمكين المتدربين من تطوير قدراتهم الفكرية على غرار المبرمجين المحترفين، والتعاون بكفاءة ضمن فريق محترف في شركة "جمال تك" أو أي مؤسسة متعددة الجنسيات أخرى. نظرًا لأهمية اللغة الإنجليزية في بيئة العمل العالمية، يتم تقديم المحتوى التدريبي بالإنجليزية. لا يشترط إتقان اللغة بشكل كامل، لكن من الضروري امتلاك القدرة الكافية لفهم المتطلبات وتنفيذها بشكل فعّال. يُمكن للمتدربين استخدام مترجم جوجل أو الاستعانة بـ "شات جي بي تي" للتغلب على أية عقبات لغوية، المهم هو الفهم الدقيق للمطلوب وتحقيقه بنجاح.

لتعظيم الاستفادة من التدريب، يُنصح بمحاولة حل التمارين بشكل مستقل لمدة ساعة واحدة على الأقل قبل الرجوع إلى الحل المرفق في نهاية الملف.

قد يتضمن الحل كودًا برمجيًا غير مفسر بعد، والغرض من ذلك هو تشجيعك على محاولة فهم الأكواد البرمجية الجديدة التي لم تتعرض لها من قبل. هذه المهارة ضرورية في سوق العمل، حيث تتطور لغات البرمجة باستمرار ويظهر كل يوم لغات جديدة. ستواجه دائمًا أكوادًا لم تدرسها من قبل، ومن المهم أن تكون قادرًا على فهمها بنفسك دون الحاجة إلى دراسة مسبقة. يمكنك الاستعانة بمحرك البحث جوجل، أو استخدام ChatGPT، أو حتى اللجوء لأصدقائك للمساعدة. الهدف الأساسي هو أن تصل إلى فهم معنى كل كود بأي طريقة ممكنة لتتمكن من إيجاد موقعك في سوق العمل.

إن وجود كود برمجي غير مفسر يشكل تحديًا يتوجب عليك إيجاد حل له. هذا النوع من التدريبات يعد جزءًا أساسيًا من تدريبات 'Problem Solving'، التي تهدف إلى تمكينك من أداء عملك بفاعلية بغض النظر عن التحديات والعقبات. هذه القدرة على حل المشكلات هي ما يتمتع به العاملون في 'جمال تك'، ومن الضروري أن تطور في نفسك هذه المهارة لتصبح عضوًا فعالًا في فريق عمل 'جمال تك'.

Eco-Sync at Gammal Tech

Background:

Gammal Tech, a pioneering software development company, has embarked on a project named "Eco-Sync" to save the planet. Using their expertise in data structures, the team at Gammal Tech developed an innovative system to monitor and balance ecological parameters across various global locations.

Problem Statement:

The Eco-Sync system uses a double circular linked list to store data points representing different ecological parameters from various global locations. Each node in the list contains two values: `locationID` (a unique integer identifier for a location) and `ecoValue` (an integer representing an ecological parameter value for that location).

Your task is to write a program that processes a series of commands to manipulate this linked list and provides outputs according to the operations performed.

Commands:

- `ADD locationID ecoValue`: Adds a new node to the list with the given `locationID` and `ecoValue`. If the `locationID` already exists, update the `ecoValue`.
- `REMOVE locationID`: Removes the node with the given `locationID` from the list.
- `PRINT`: Prints the `locationID` and `ecoValue` of all nodes in the list, starting from the lowest `locationID` and following the next pointer in the circular list.

Constraints:

- `1 ≤ locationID ≤ 10^6`
- `-10^6 ≤ ecoValue ≤ 10^6`
- Number of commands `≤ 10^4`

Input Format:

- The first line contains an integer `N`, the number of commands to follow.
- The next `N` lines each contain a command (`ADD`, `REMOVE`, or `PRINT`).

Output Format:

- For each `PRINT` command, output a line for each node in the list, in the format `locationID ecoValue`. If the list is empty, print `EMPTY`.

Sample Input

```
5
ADD 101 300
ADD 102 400
PRINT
REMOVE 101
PRINT
```

Sample Output

```
101 300
102 400
102 400
```

لتعظيم الاستفادة من التدريب، يُنصح بمحاولة حل التمرين بشكل مستقل لمدة ساعة واحدة على الأقل قبل الرجوع إلى الحل المرفق

C++ Solution:

```
#include <iostream>
#include <map>
using namespace std;

struct Node {
    int locationID;
    int ecoValue;
    Node *next, *prev;
    Node(int id, int value) : locationID(id), ecoValue(value),
next(nullptr), prev(nullptr) {}
};

class EcoList {
    Node *head;
    map<int, Node*> nodeMap;

public:
    EcoList() : head(nullptr) {}

    void addNode(int locationID, int ecoValue) {
        if (nodeMap.find(locationID) != nodeMap.end()) {
            // Update ecoValue if locationID exists
            nodeMap[locationID]->ecoValue = ecoValue;
            return;
        }

        Node *newNode = new Node(locationID, ecoValue);
        if (!head) {
            head = newNode;
            head->next = head->prev = head;
        } else {
            // Insert in sorted order
            Node *curr = head;
            while (curr->next != head && curr->next->locationID <
locationID)
                curr = curr->next;

            newNode->next = curr->next;
            newNode->prev = curr;
            curr->next->prev = newNode;
            curr->next = newNode;
            if (curr == head && locationID < head->locationID)
                head = newNode; // Update head for lower locationID
        }
        nodeMap[locationID] = newNode;
    }

    void removeNode(int locationID) {
        if (nodeMap.find(locationID) == nodeMap.end()) return;
```

```
        Node *delNode = nodeMap[locationID];
        if (delNode == head && head->next == head) {
            head = nullptr;
        } else {
            delNode->prev->next = delNode->next;
            delNode->next->prev = delNode->prev;
            if (delNode == head) head = delNode->next;
        }

        nodeMap.erase(locationID);
        delete delNode;
    }

    void printList() {
        if (!head) {
            cout << "EMPTY\n";
            return;
        }
        Node *curr = head;
        do {
            cout << curr->locationID << " " << curr->ecoValue << "\n";
            curr = curr->next;
        } while (curr != head);
    }
};

int main() {
    int N;
    cin >> N;
    EcoList ecoList;

    while (N--) {
        string command;
        cin >> command;

        if (command == "ADD") {
            int id, value;
            cin >> id >> value;
            ecoList.addNode(id, value);
        } else if (command == "REMOVE") {
            int id;
            cin >> id;
            ecoList.removeNode(id);
        } else if (command == "PRINT") {
            ecoList.printList();
        }
    }

    return 0;
}
```