Problem Solving (C33)

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'.

Gammal Tech's Dynamic Network Allocator

Background:

Gammal Tech, a trailblazer in the software development industry, is renowned for its innovative approaches and state-of-the-art technologies. They are now developing a new network system that dynamically allocates resources to optimize workflow and reduce latency in their high-tech office environment.

Problem Statement:

You are part of Gammal Tech's elite software team, tasked with developing a part of the Dynamic Network Allocator (DNA). The DNA system must manage a series of network nodes. Each node can be either active or inactive and can dynamically change its state based on resource demand.

Your task is to design a system using C programming that:

1. Initializes an array of `N` pointers, each pointing to a network node. Each node contains its `id` (a unique integer), `status` (an integer where 0 represents inactive, and 1 represents active), and a `load` (an integer representing the current load).
2. Implements a function `toggleNodeStatus` which takes the node `id` and toggles its status (from active to inactive or vice versa).
3. Implements a function `updateNodeLoad` which updates the `load` of a node given its `id` and the new `load` value.
4. Implements a function `displayActiveNodes` which prints the `id` and `load` of all active nodes.

Constraints:

- 1 ≤ N ≤ 1000
- 0 ≤ load ≤ 10000
- Node `id`s are unique and range from 0 to N-1

Input Format

- The number of nodes, N
- A series of commands (either toggle the status of a node, update the load of a node, or display active nodes).

Output Format

- For each 'displayActiveNodes' command, output the `id` and `load` of all active nodes in ascending order of their `id`.

Sample Input:

```
5
toggleNodeStatus 3
updateNodeLoad 3 500
displayActiveNodes
toggleNodeStatus 2
updateNodeLoad 2 300
displayActiveNodes
```

Sample Output:

```
3 500
2 300
3 500
```

لتحقيق أقصى فائدة من التدريب، يُوصى ببذل محاولة مستقلة لحل التمارين لمدة لا تقل عن ساعة واحدة. تجنب الاطلاع على بقية الملف حتى تكمل عملية التفكير في الحل. بعد ذلك، جرب حلك بنفسك على المدخلات الموضحة. إذا واجهت مدخلات لم تتوقعها، فهذا يعد فرصة لتطوير مهارة جديدة ضمن مسيرتك التعليمية. المهندس المحترف يجب أن يضمن أن برنامجه يعمل مع جميع أنواع المدخلات، وهذه مهارة يتم تطويرها عبر التجربة والخطأ. لذا، من الضروري ألا تطلع على المدخلات المتوقعة قبل أن تجرب الحل بنفسك. هذه هي الطريقة الأمثل لتنمية هذه المهارة.

بعد اختبار المدخلات المقترحة، إذا كانت النتائج تختلف عما هو مدون في الملف، فيُنصح بمحاولة حل التمرين مرة أخرى لمدة ساعة على الأقل قبل الرجوع إلى الحل الموجود في نهاية الملف.

Test Case 1: No Nodes Active

Tests the system's behavior when no nodes are active.

Input

```
5
displayActiveNodes
```

Expected Output

```

```

Test Case 2: All Nodes Active and Updated

Tests the system's response when all nodes are active and have updated loads.

Input

```
3
toggleNodeStatus 0
updateNodeLoad 0 100
toggleNodeStatus 1
updateNodeLoad 1 200
toggleNodeStatus 2
updateNodeLoad 2 300
displayActiveNodes
```

Expected Output

```
0 100
1 200
2 300
```

Test Case 3: Toggle Same Node Multiple Times

Tests how the system handles multiple status toggles for the same node.

Input

```
2
toggleNodeStatus 1
toggleNodeStatus 1
toggleNodeStatus 1
updateNodeLoad 1 400
displayActiveNodes
```

Expected Output

```
1 400
```

Test Case 4: Large Number of Nodes

Tests system performance and correctness with a large number of nodes.

Input

```
1000
toggleNodeStatus 999
updateNodeLoad 999 9999
displayActiveNodes
```

Expected Output

```
999 9999
```

Test Case 5: Update Load of Inactive Node

Checks if updating the load of an inactive node affects the output.

Input

```
4
updateNodeLoad 2 500
displayActiveNodes
```

Expected Output

```

```

لتحقيق أقصى استفادة من التدريب، من المستحسن أن تخصص وقتًا إضافيًا - لا يقل عن ساعة - لمحاولة حل التمرين مرة أخرى بمفردك قبل الرجوع إلى الحل المقترح. هذه العملية المتكررة من التجربة والخطأ تعتبر استراتيجية فعالة في تعزيز مهاراتك البرمجية وتعميق فهمك للمفاهيم. تذكر أن التحدي والمثابرة هما المفتاحان للتطور في مجال البرمجة.

C Programming Solution:

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

typedef struct {
    int id;
    int status;
    int load;
} Node;

void toggleNodeStatus(Node *nodes[], int id) {
    nodes[id]->status = !nodes[id]->status;
}

void updateNodeLoad(Node *nodes[], int id, int newLoad) {
    nodes[id]->load = newLoad;
}

void displayActiveNodes(Node *nodes[], int n) {
    for (int i = 0; i < n; i++) {
        if (nodes[i]->status == 1) {
            printf("%d %d\n", nodes[i]->id, nodes[i]->load);
        }
    }
}

int main() {
    int n;
    scanf("%d", &n);

    Node *nodes[n];
    for (int i = 0; i < n; i++) {
        nodes[i] = (Node *)malloc(sizeof(Node));
        nodes[i]->id = i;
        nodes[i]->status = 0; // Initially, all nodes are inactive
        nodes[i]->load = 0;
    }

    // Example Commands
    toggleNodeStatus(nodes, 3);
    updateNodeLoad(nodes, 3, 500);
    displayActiveNodes(nodes, n);
    toggleNodeStatus(nodes, 2);
    updateNodeLoad(nodes, 2, 300);
    displayActiveNodes(nodes, n);

    // Free allocated memory
    for (int i = 0; i < n; i++) {
        free(nodes[i]);
    }
```

```
    return 0;
}
```