Problem Solving (C32)

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 Smart Warehouse System

Scenario:

Gammal Tech, a leading software development company, is renowned for its innovative solutions and state-of-the-art facilities. They are now tackling a critical industry issue in warehouse management. Their newest project is to develop a smart warehouse system to optimize storage efficiency and retrieval processes.

Problem Statement:

Gammal Tech's warehouse has `N` storage units, each capable of storing different types of items. The challenge is to design a system to keep track of the items in each unit efficiently. The system should be able to:

Add an item: Record a new item in the warehouse with its details.
Find an item: Quickly retrieve information about any item.
Update storage: Modify details of items as they are moved or used.

Each item has the following attributes:

- `ID`: A unique identifier.
- `Type`: The type of the item (e.g., electronics, clothing).
- `Quantity`: Number of items in the unit.
- `Location`: The storage unit number where the item is located.

Task

Implement the warehouse system using an array of structs in C. The program should be able to process a series of commands to add, find, and update items in the warehouse. The commands are:

- `A` (Add): Followed by item details to add a new item.
- `F` (Find): Followed by an item ID to find its details.
- `U` (Update): Followed by item ID and new details to update the item.

Input Format

- The first line contains an integer `N`, the number of commands.
- The next `N` lines contain commands (`A`, `F`, `U`) followed by their respective details.

Output Format

- For each `Find` command, output the details of the item. If the item is not found, print "Not Found".

Constraints:

- 1 ≤ `N` ≤ 10^5
- 1 ≤ `ID` ≤ 10^6
- `Type` is a string with a maximum length of 50 characters.
- 1 ≤ `Quantity` ≤ 10^3
- 1 ≤ `Location` ≤ `N`

Sample Input:

```
5
A 123 Electronics 50 1
A 456 Clothing 20 2
F 123
U 456 Clothing 15 3
F 456
```

Sample Output:

```
123 Electronics 50 1
456 Clothing 15 3
```

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: Large Quantity and Location at Upper Bound

Input

```
2
A 999999 Electronics 1000 100000
F 999999
```

Expected Output

```
999999 Electronics 1000 100000
```

Purpose: This test checks if the system can handle the maximum limits for `Quantity` and `Location`.

Test Case 2: Item Not Found

Input

```
2
A 100 Clothing 20 5
F 200
```

Expected Output

```
Not Found
```

Purpose: This test ensures that the system correctly outputs "Not Found" for an item ID that does not exist.

Test Case 3: Updating an Item That Doesn't Exist

Input

```
2
A 300 Electronics 50 3
U 400 Electronics 60 4
```

Expected Output

```

```

Purpose: This test checks the system's behavior when trying to update an item that hasn't been added. It should handle this silently without any output or errors.

Test Case 4: Multiple Operations on the Same Item

Input

```
4
A 123 Electronics 50 1
F 123
U 123 Electronics 30 2
F 123
```

Expected Output

```
123 Electronics 50 1
123 Electronics 30 2
```

Purpose: To verify that the system accurately handles multiple operations (add, find, update) on the same item.

Test Case 5: Maximum Number of Items

Input

```
3
A 1 Food 10 1
A 2 Clothes 15 2
A 3 Electronics 5 3
```

Expected Output

```

```

Purpose: This test is to ensure that the system can handle the maximum number of items (`N`) without any issues. The input does not require any output but is meant to test the system's capacity handling.

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 <string.h>

typedef struct {
    int ID;
    char Type[51];
    int Quantity;
    int Location;
} Item;

void addItem(Item *warehouse, int *count, int ID, char *Type, int
Quantity, int Location) {
    warehouse[*count].ID = ID;
    strcpy(warehouse[*count].Type, Type);
    warehouse[*count].Quantity = Quantity;
    warehouse[*count].Location = Location;
    (*count)++;
}

int findItem(Item *warehouse, int count, int ID) {
    for (int i = 0; i < count; i++) {
        if (warehouse[i].ID == ID) {
            return i;
        }
    }
    return -1;
}

void updateItem(Item *warehouse, int index, char *Type, int Quantity,
int Location) {
    strcpy(warehouse[index].Type, Type);
    warehouse[index].Quantity = Quantity;
    warehouse[index].Location = Location;
}

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

    Item warehouse[N];
    int count = 0;

    for (int i = 0; i < N; i++) {
        char command;
        int ID, Quantity, Location;
        char Type[51];

        scanf(" %c", &command);

        if (command == 'A') {
```

```
            scanf("%d %s %d %d", &ID, Type, &Quantity, &Location);
            addItem(warehouse, &count, ID, Type, Quantity, Location);
        } else if (command == 'F') {
            scanf("%d", &ID);
            int index = findItem(warehouse, count, ID);
            if (index != -1) {
                printf("%d %s %d %d\n", warehouse[index].ID,
warehouse[index].Type, warehouse[index].Quantity,
warehouse[index].Location);
            } else {
                printf("Not Found\n");
            }
        } else if (command == 'U') {
            scanf("%d %s %d %d", &ID, Type, &Quantity, &Location);
            int index = findItem(warehouse, count, ID);
            if (index != -1) {
                updateItem(warehouse, index, Type, Quantity, Location);
            }
        }
    }

    return 0;
}
```