Problem Solving (C59)

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 Database Design Challenge

Scenario:

Gammal Tech, a leader in the software development industry known for its cutting-edge technology and innovative approach, is embarking on a new project. The company is tasked with designing a simplified relational database system for a small business. The challenge is to create a system that efficiently manages employee and department data, showcasing Gammal Tech's excellence in system design and understanding of relational databases.

Problem Statement:

Your task is to write a program in C that simulates the management of two relational database tables: `Employees` and `Departments`. Each employee is associated with exactly one department, but each department can have multiple employees (one-to-many relationship). Additionally, there is a special case of a one-to-one relationship between departments and their managers (who are also employees).

- Employees Table: Contains employee ID, name, and department ID.

- Departments Table: Contains department ID, department name, and manager ID.

Your program should be able to process queries to:

Add a new employee.
Add a new department.
Assign a manager to a department (ensuring the manager is an employee).
Retrieve all employees in a given department.

Input Format:

- The first line contains an integer `Q`, the number of queries.
- The following `Q` lines describe each query, which can be of four types:
  - `1 <employee_id> <employee_name> <department_id>`: Add a new employee.
  - `2 <department_id> <department_name>`: Add a new department.
  - `3 <department_id> <manager_id>`: Assign a manager to a department.
  - `4 <department_id>`: Retrieve all employees in the specified department.

Output Format:

- For each type 4 query, output a single line containing the names of employees in the specified department, separated by commas. If the department is empty, output `No employees`.

Sample Input:

```
5
1 101 John 201
1 102 Alice 202
2 201 Sales
2 202 Marketing
4 201
```

Sample Output:

```
John
```

Constraints:

- 1 ≤ Q ≤ 100
- Employee and department IDs are integers.
- Names are strings without spaces.

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

C Programming Solution:

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

typedef struct {
    int id;
    char name[50];
    int dept_id;
} Employee;

typedef struct {
    int id;
    char name[50];
    int manager_id;
} Department;

Employee employees[100];
Department departments[100];
int emp_count = 0, dept_count = 0;

void addEmployee(int id, char *name, int dept_id) {
    employees[emp_count].id = id;
    strcpy(employees[emp_count].name, name);
    employees[emp_count].dept_id = dept_id;
    emp_count++;
}

void addDepartment(int id, char *name) {
    departments[dept_count].id = id;
    strcpy(departments[dept_count].name, name);
    departments[dept_count].manager_id = -1;
    dept_count++;
}
```

```c
void assignManager(int dept_id, int manager_id) {
    for (int i = 0; i < dept_count; i++) {
        if (departments[i].id == dept_id) {
            departments[i].manager_id = manager_id;
            break;
        }
    }
}

void listEmployees(int dept_id) {
    int found = 0;
    for (int i = 0; i < emp_count; i++) {
        if (employees[i].dept_id == dept_id) {
            if (found > 0) printf(", ");
            printf("%s", employees[i].name);
            found++;
        }
    }
    if (found == 0) printf("No employees");
    printf("\n");
}

int main() {
    int Q, query_type, id, dept_id;
    char name[50];

    scanf("%d", &Q);
    for (int i = 0; i < Q; i++) {
        scanf("%d", &query_type);
        switch (query_type) {
            case 1:
                scanf("%d %s %d", &id, name, &dept_id);
                addEmployee(id, name, dept_id);
                break;
            case 2:
                scanf("%d %s", &id, name);
                addDepartment(id, name);
                break;
            case 3:
                scanf("%d %d", &dept_id, &id);
                assignManager(dept_id, id);
                break;
            case 4:
                scanf("%d", &dept_id);
                listEmployees(dept_id);
                break;
        }
    }
    return 0;
}
```