Problem Solving (C69)

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 Header File Harmony

Background:
Gammal Tech, a leader in software development, is developing a new technology that revolutionizes how header files are managed in large-scale software projects. This innovative system, named HeaderSync, ensures a harmonized and conflict-free integration of multiple header files, showcasing Gammal Tech's commitment to excellence and innovation in the software industry.

Problem Statement:
Your task is to write a program for Gammal Tech that reads a list of header files used in a software project and detects any potential conflicts. A conflict occurs if two or more header files define the same macro with different values. Your program should identify these conflicts to assist developers in maintaining a consistent and error-free codebase.

Input Format:

- The first line contains an integer `N`, the number of header files.

- The next `N` lines each contain a string representing the header file name followed by an integer `M`, the number of macros defined in that header file, followed by `M` pairs of strings and integers representing the macro name and its value.

Output Format:

- For each conflicting macro, print the macro name followed by a list of header files where the conflict occurs.
- If no conflicts are found, print `"No conflicts detected."`

Sample Input:

```
3
file1.h 2 MAX_SIZE 100 MIN_SIZE 50
file2.h 2 MAX_SIZE 100 MAX_VALUE 200
file3.h 1 MAX_SIZE 150
```

Sample Output:

```
MAX_SIZE: file1.h, file3.h
```

لتعظيم 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 <stdlib.h>
#include <string.h>

typedef struct {
    char macroName[50];
    int value;
} Macro;

typedef struct {
    char fileName[50];
    int macroCount;
    Macro *macros;
} HeaderFile;
```

```c
int main() {
    int n, i, j, k;
    scanf("%d", &n);

    HeaderFile *files = malloc(n * sizeof(HeaderFile));

    for (i = 0; i < n; i++) {
        scanf("%s %d", files[i].fileName, &files[i].macroCount);
        files[i].macros = malloc(files[i].macroCount * sizeof(Macro));

        for (j = 0; j < files[i].macroCount; j++) {
            scanf("%s %d", files[i].macros[j].macroName,
&files[i].macros[j].value);
        }
    }

    int conflictFound = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < files[i].macroCount; j++) {
            for (k = i + 1; k < n; k++) {
                for (int l = 0; l < files[k].macroCount; l++) {
                    if (strcmp(files[i].macros[j].macroName,
files[k].macros[l].macroName) == 0 &&
                        files[i].macros[j].value !=
files[k].macros[l].value) {
                        if (!conflictFound) {
                            printf("%s: %s, ",
files[i].macros[j].macroName, files[i].fileName);
                            conflictFound = 1;
                        }
                        printf("%s, ", files[k].fileName);
                    }
                }
            }
            if (conflictFound) {
                printf("\b\b \n");
                conflictFound = 0;
            }
        }
    }

    if (!conflictFound) {
        printf("No conflicts detected.\n");
    }

    for (i = 0; i < n; i++) {
        free(files[i].macros);
    }
    free(files);
    return 0;
}
```