Problem Solving (CPP26)

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 String Formatter

Background

Gammal Tech, a leading software development company renowned for its innovative solutions, exceptional work environment, and state-of-the-art facilities, is embarking on a mission to teach its interns the fundamentals of system design with a focus on C++ classes. To facilitate this, Gammal Tech has designed a unique problem that not only tests the interns' understanding of string manipulation in C++ but also instills best practices in system design.

Problem Statement

Gammal Tech needs a C++ program that can process a series of textual commands to format strings. Each command will instruct the program to perform a specific operation on the string, such as reversing the string, changing the case of letters, or concatenating a new string. The program must be designed using classes to showcase the power of object-oriented programming in system design.

Objectives

- Design a `StringFormatter` class that can execute commands like `reverse`, `toUpper`, `toLower`, and `concatenate`.
- The class should maintain the state of the string after each operation.
- Implement an efficient way to process a series of commands on the string.

Input Format

- The first line contains the initial string `S`.
- The second line contains an integer `N`, the number of commands to follow.
- The next `N` lines each contain a command. Each command is one of the following:
  - `reverse`
  - `toUpper`
  - `toLower`
  - `concatenate <string>`

Output Format

- After processing all commands, output the final state of the string.

Constraints

- $1 \leq |S| \leq 1000$ (where $|S|$ is the length of the initial string)
- $1 \leq N \leq 100$
- Strings will contain only ASCII characters.

Sample Input:

```
HelloWorld
3
reverse
toUpper
concatenate Tech
```

Sample Output:

```
DLROWOLLEHTech
```

Explanation

The initial string is "HelloWorld". The first command reverses it to "dlroWolleH". The second command changes it to uppercase "DLROWOLLEH". The final command concatenates "Tech" resulting in "DLROWOLLEHTech".

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

C++ Programming Solution:

```cpp
#include <iostream>
#include <algorithm>
#include <string>

class StringFormatter {
private:
    std::string str;

public:
    StringFormatter(const std::string &initialStr) : str(initialStr) {}

    void reverse() {
        std::reverse(str.begin(), str.end());
    }

    void toUpper() {
        std::transform(str.begin(), str.end(), str.begin(),
            [](unsigned char c){ return std::toupper(c); });
    }

    void toLower() {
        std::transform(str.begin(), str.end(), str.begin(),
            [](unsigned char c){ return std::tolower(c); });
    }

    void concatenate(const std::string &additionalStr) {
        str += additionalStr;
    }

    std::string getResult() {
        return str;
    }
};
```

```
int main() {
    std::string initialStr, command, extra;
    int numOfCommands;

    std::getline(std::cin, initialStr);
    std::cin >> numOfCommands;
    std::cin.ignore();  // Ignore the newline after reading
numOfCommands

    StringFormatter formatter(initialStr);

    for (int i = 0; i < numOfCommands; i++) {
        std::getline(std::cin, command);
        if (command == "reverse") {
            formatter.reverse();
        } else if (command == "toUpper") {
            formatter.toUpper();
        } else if (command == "toLower") {
            formatter.toLower();
        } else if (command.substr(0, 11) == "concatenate") {
            extra = command.substr(12);
            formatter.concatenate(extra);
        }
    }

    std::cout << formatter.getResult() << std::endl;
    return 0;
}
```