Problem Solving (CPP24)

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'.

Design Patterns at Gammal Tech

Background:

Gammal Tech, a leading software development company known for its cutting-edge technology and innovative solutions, is hosting an internship program. As part of their training, the interns are given a challenge to understand and apply design patterns in system design.

Scenario:

The interns at Gammal Tech have been tasked with designing a component of a new software system that is modular, efficient, and easily maintainable. The system is part of Gammal Tech's latest project, an AI-powered analytics tool designed to predict market trends. Your task is to use an appropriate design pattern to create a module that seamlessly integrates with the existing system.

Problem:

Create a C++ program using a suitable design pattern that accomplishes the following:

Module Integration: Your module must interact with existing system components like Database Access, User Interface, and Data Processing units.
Scalability and Maintenance: The design should be scalable and easy to maintain.
Flexibility: The module should be adaptable to potential future changes in the system, such as additional features or integration with other tools.

Design Pattern Requirement:

You must choose and implement one of the following design patterns in your solution:

- Singleton Pattern
- Observer Pattern
- Factory Method Pattern

Input Format:

- The first line contains an integer `N`, the number of operations to be performed.
- The next `N` lines describe the operations. Each line starts with a string indicating the operation type followed by necessary parameters separated by spaces.

Output Format:

For each operation, output a line describing the result or status after the operation is executed. The format will depend on the operation type.

Sample Input:

```
3
CreateUI Window
UpdateDB EmployeeData
ProcessData MarketAnalysis
```

Sample Output:

```
UI Component Window created successfully.
Database EmployeeData updated successfully.
Data Processing for MarketAnalysis completed.
```

Constraints:

- $1 \leq N \leq 1000$
- Operations are limited to 'CreateUI', 'UpdateDB', and 'ProcessData'.

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 <string>

// Abstract Product
class Component {
public:
    virtual void operate() = 0;
};

// Concrete Products
class UIComponent : public Component {
    std::string uiType;
public:
    UIComponent(std::string type) : uiType(type) {}
    void operate() override {
        std::cout << "UI Component " << uiType << " created
successfully.\n";
    }
};

class DBComponent : public Component {
    std::string dbName;
public:
    DBComponent(std::string name) : dbName(name) {}
    void operate() override {
        std::cout << "Database " << dbName << " updated
successfully.\n";
    }
};
```

```
class DataProcessingComponent : public Component {
    std::string dataType;
public:
    DataProcessingComponent(std::string type) : dataType(type) {}
    void operate() override {
        std::cout << "Data Processing for " << dataType << "
completed.\n";
    }
};

// Creator
class ComponentFactory {
public:
    static Component* createComponent(const std::string& type, const
std::string& param) {
        if (type == "CreateUI") {
            return new UIComponent(param);
        } else if (type == "UpdateDB") {
            return new DBComponent(param);
        } else if (type == "ProcessData") {
            return new DataProcessingComponent(param);
        }
        return nullptr;
    }
};

int main() {
    int N;
    std::string operation, param;
    std::cin >> N;
    for (int i = 0; i < N; i++) {
        std::cin >> operation >> param;
        Component* component =
ComponentFactory::createComponent(operation, param);
        if (component) {
            component->operate();
            delete component;
        }
    }
    return 0;
}
```