

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І. І. МЕЧНИКОВА
ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ ТА ІНФОРМАЦІЙНИЙ ТЕХНОЛОГІЙ
КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

Лабораторна робота №8
з дисципліни: «Інженерія програмного забезпечення»
за темою: «Патерн проектування Iterator»

Виконав:
студент 3 курсу
Владислав Краковський

Перевірив:
викладач
Пенко В.Г.

Зміст

1 ЗАВДАННЯ РОБОТИ.....	3
2 ХІД РОБОТИ.....	4
Висновок.....	10

1 ЗАВДАННЯ РОБОТИ

Мета роботи: Ознайомитися з патерном проєктування “Ітератор” (Iterator), навчитися реалізовувати його на мові програмування C++, зрозуміти його основні переваги, недоліки й сфери застосування.

Умова роботи:

Варіанти самостійних завдань

1. Агрегат містить 2 списки слів – список англійських слів та їх переклад українською мовою. Визначивши кілька ітераторів, вивести за їх допомогою:

- – список лише англійських слів;
- – список лише українських слів;
- – список пар <англійське слово-українське слово>.

Рисунок 1.1 - Умова

Система контролю версій Git та рефакторинг

Варіанти самостійних завдань

У наведеному вище викладі кожного патерну наводяться структурний та реальний приклад. Реалізувати структурний приклад та зафіксувати його в системі контролю версій. Далі здійснити серію фіксацій, яка трансформує структурний приклад у реальний. Деякі з проміжних фіксацій повинні полягати у застосуванні того чи іншого прийому рефакторингу.

Рисунок 1.2 - Завдання з GitHub

2 ХІД РОБОТИ

Опис патерну Iterator:

Iterator (Ітератор) — це поведінковий патерн проєктування, який дозволяє послідовно перебирати елементи колекції (наприклад, масиву, списку, дерева) без розкриття її внутрішньої реалізації.

Патерн надає уніфікований інтерфейс для проходження по елементах структури даних.

Як реалізується патерн Iterator?

1. Виділяється інтерфейс ітератора (абстрактний клас), який визначає базові операції для перебору елементів: отримання першого елемента, перехід до наступного, перевірка кінця, отримання поточного елемента.
2. Колекція (Aggregate) має метод для створення ітератора для своїх елементів.
3. Реалізуються конкретні ітератори для різних колекцій (масив, список, дерево тощо), які працюють згідно з цим інтерфейсом.
4. Клієнт використовує ітератор для перебору колекції, не знаючи її внутрішню структуру.

У C++ це зазвичай — створення класу-ітератора із методами First(), Next(), IsDone(), CurrentItem() та класу-колекції, що створює ітератор.

Для чого використовується патерн **Iterator**?

- Для організації послідовного доступу до елементів колекцій різного типу.
- Щоб уніфікувати обхід різних структур даних у коді.
- Для спрощення клієнтського коду (клієнт не залежить від типу колекції).

Типові приклади:

- Перебір елементів списку, дерева, графа.
- Реалізація циклів `foreach` у мовах програмування.
- Робота з контейнерами STL у C++ (`std::vector`, `std::list`, `std::map`).

Переваги патерну **Iterator**

- Ізоляція перебору від структури даних — колекцію можна змінювати, не змінюючи ітератор.
- Єдиний інтерфейс перебору для різних типів колекцій.
- Можна створювати декілька ітераторів для однієї колекції (наприклад, прямий і зворотній перебір).

Недоліки патерну **Iterator**

- Додає нові класи (інтерфейс ітератора, конкретний ітератор).
- Не завжди ефективний для складних або динамічно змінних структур (наприклад, при видаленні елементів під час перебору).
- Підвищує складність коду при великій кількості різних ітераторів.

Запишемо розв'язок поставленої задачі з використання патерну Iterator:

```
#include <iostream>
#include <vector>
#include <string>

// Клас-агрегат для двох списків слів
class WordPairsAggregate {
    std::vector<std::string> englishWords;
    std::vector<std::string> ukrainianWords;
public:
    WordPairsAggregate(const std::vector<std::string>& en, const
std::vector<std::string>& ua)
        : englishWords(en), ukrainianWords(ua) {}

    int Count() const { return englishWords.size(); }

    const std::string& GetEnglish(int i) const { return
englishWords.at(i); }
    const std::string& GetUkrainian(int i) const { return
ukrainianWords.at(i); }
};

// Ітератор по англійським словам
class EnglishIterator {
    const WordPairsAggregate& agg;
    int current;
public:
    EnglishIterator(const WordPairsAggregate& a) : agg(a),
current(0) {}
    void First() { current = 0; }
    void Next() { ++current; }
    bool IsDone() const { return current >= agg.Count(); }
    std::string CurrentItem() const { return
agg.GetEnglish(current); }
```

```
};
```

```
// Итератор по украинским словам
```

```
class UkrainianIterator {
    const WordPairsAggregate& agg;
    int current;
public:
    UkrainianIterator(const WordPairsAggregate& a) : agg(a),
current(0) {}
    void First() { current = 0; }
    void Next() { ++current; }
    bool IsDone() const { return current >= agg.Count(); }
    std::string CurrentItem() const { return
agg.GetUkrainian(current); }
};
```

```
// Итератор по парам слов
```

```
class PairIterator {
    const WordPairsAggregate& agg;
    int current;
public:
    PairIterator(const WordPairsAggregate& a) : agg(a), current(0)
{}
    void First() { current = 0; }
    void Next() { ++current; }
    bool IsDone() const { return current >= agg.Count(); }
    std::pair<std::string, std::string> CurrentItem() const {
        return { agg.GetEnglish(current),
agg.GetUkrainian(current) };
    }
};
```

```
int main() {
    setlocale(LC_ALL, "");
```

```

    std::vector<std::string> en = { "cat", "dog", "apple",
    "book" };

    std::vector<std::string> ua = { "кіт", "пес", "яблуко",
    "книга" };

    WordPairsAggregate dict(en, ua);

    // Английские слова
    std::cout << "Список английских слов:\n";
    EnglishIterator it_en(dict);
    for (it_en.First(); !it_en.IsDone(); it_en.Next())
        std::cout << it_en.CurrentItem() << std::endl;

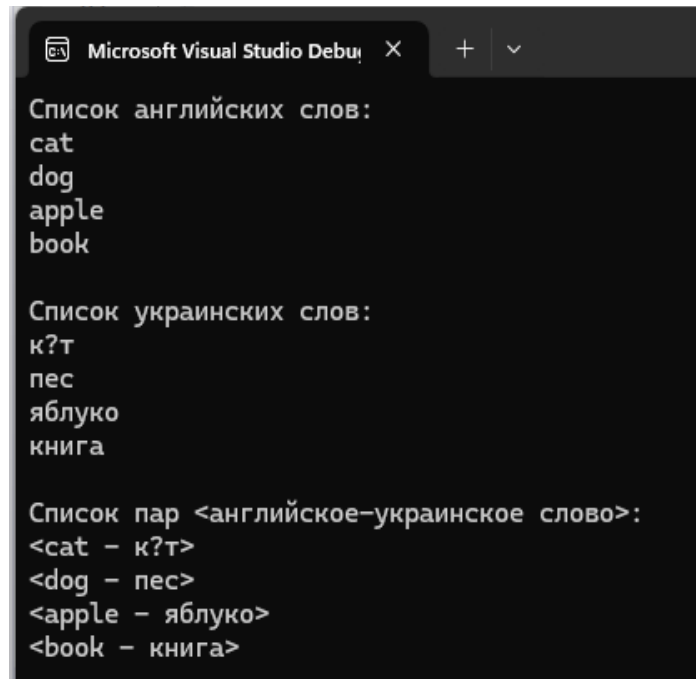
    // Украинские слова
    std::cout << "\nСписок украинских слов:\n";
    UkrainianIterator it_ua(dict);
    for (it_ua.First(); !it_ua.IsDone(); it_ua.Next())
        std::cout << it_ua.CurrentItem() << std::endl;

    // Пары слов
    std::cout << "\nСписок пар <английское-украинское слово>:\n";
    PairIterator it_pair(dict);
    for (it_pair.First(); !it_pair.IsDone(); it_pair.Next()) {
        auto pair = it_pair.CurrentItem();
        std::cout << "<" << pair.first << " - " << pair.second <<
">" << std::endl;
    }

    return 0;
}

```


Подивимось скріншот з результатом на ілюстрації 2.1:



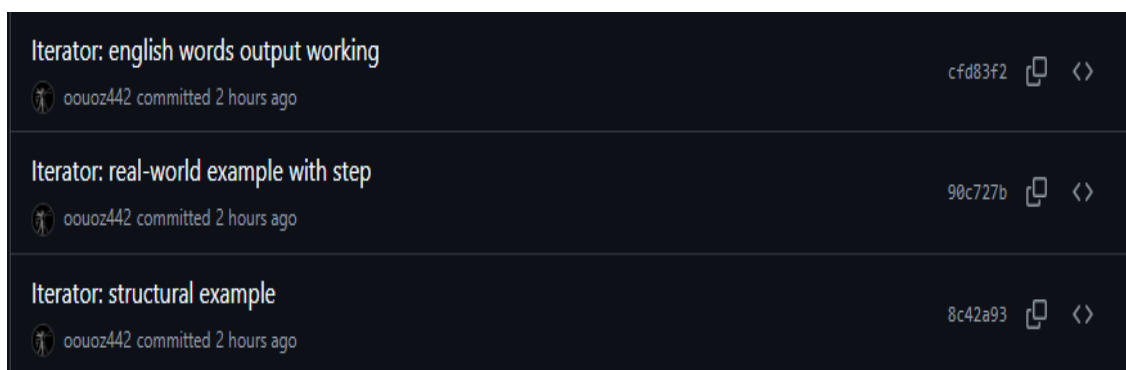
```
Microsoft Visual Studio Debug Console
Список английских слов:
cat
dog
apple
book

Список украинских слов:
к?т
пес
яблуко
книга

Список пар <английское-украинское слово>:
<cat - к?т>
<dog - пес>
<apple - яблуко>
<book - книга>
```

Рисунок 2.1 — Результат роботи програми

Програма була завантажена на GitHub згідно з завданням, ілюстрація 2.2:



Iterator: english words output working	cfd83f2	📄	<>
oouoz442 committed 2 hours ago			
Iterator: real-world example with step	90c727b	📄	<>
oouoz442 committed 2 hours ago			
Iterator: structural example	8c42a93	📄	<>
oouoz442 committed 2 hours ago			

Рисунок 2.2 - GitHub

Висновок

Патерн “Iterator” — це універсальний інструмент для організації перебору елементів у колекціях незалежно від їхньої реалізації. Він спрощує клієнтський код і підвищує модульність системи, але може призводити до збільшення кількості класів та певного ускладнення архітектури. Використання Iterator доцільне у складних або універсальних програмах, де потрібна робота з багатьма типами колекцій.