**Що таке Code Review**

**Code Review** - це перевірка коду на помилки, неточності та загальний стиль програмування.

**Ситуація** : ви розробник. Ви відповідаєте за свій фронт роботи, наприклад, за надсилання даних на сервер. У команди вже є готовий проект, ви разом його підтримуєте та постійно покращуєте.

Нова функція не потрапляє у проект. Натомість ваш код відправляється на Code Review .

**Основні принципи написання коректного коду на C ++**

1. **Мінімізуйте глобальні та спільно використовувані дані**

Уникайте спільного використання даних, особливо глобальних даних. Дані, що спільно використовуються, посилюють зв'язність, що призводить до зниження супроводжуваності, а найчастіше і продуктивності.

1. **Використовуйте RAII**

Ідіома С++ "виділення ресурсу є ініціалізація" (resource acquisition is initialization - RAII) є потужним інструментом для коректної роботи з ресурсами. У разі виділення ресурсу передайте його об'єкту-власнику.

1. **Активно використовуйте const**

Незмінні значення простіше розуміти, відстежувати і мотивувати, тобто. там, де це доцільно, краще використовувати константи замість змінних.

1. **Уникайте макросів**

Не використовуйте макроси в C++ без крайньої необхідності. C++ надає багатий інструментарій, такий як шаблонні функції, автоматичне виведення типів (auto, decltype), constexpr functions.

1. **Уникайте магічних чисел**

Уникайте використання у коді літеральних констант на кшталт 42 або 3.1415926. Такі константи не самоочевидні та ускладнюють супровід коду.

1. **Оголошуйте змінні як можна, можливо локальніше**

Уникайте "роздмухування" областей видимості. Змінних має бути якнайменше, а час їхнього життя - якомога коротший.

1. **Завжди ініціалізуйте змінні**

Неініціалізовані змінні - поширене джерело помилок у програмах на С та С++.

1. **Зберігайте природну семантику перевантажених операторів**

Перевантажуйте оператори тільки у разі вагомих на те підстав, і зберігайте при цьому їхню природну семантику. Якщо це виявляється складним, можливо, ви неправильно використовуєте навантаження операторів.

1. **Уникайте навантаження &&, || і , (кома)**

Вбудовані оператори &&, || і , (кома) трактуються компілятором спеціальним чином. Після навантаження вони стають звичайними функціями з дуже чудовою семантикою.

1. **Вважайте за краще мінімальні класи монолітним**

Невеликі класи легше писати, тестувати та використовувати. Вони також застосовні у більшій кількості ситуацій.

1. **Вважайте за краще композицію наслідуванню**

Сильні зв'язки небажані, їх слід уникати скрізь, де тільки можна. Слід віддавати перевагу композиції наслідуванню, крім випадків, коли ви точно знаєте, що робите і які переваги дає наслідування у вашому проекті.

1. **Уникайте наслідування від класів, які не спроектовані з цією метою**

Класи, призначені для автономного використання, підпорядковуються правилам проектування, відмінним від правил для базових класів. Використання автономних класів як базові є серйозною помилкою проектування і його слід уникати.

1. **Не допускайте втручання у внутрішні справи**

Уникайте повернення дескрипторів внутрішніх даних, керованих вашим класом, щоб клієнти не могли неконтрольовано змінювати стан вашого об'єкта як свого власного.

1. **Визначайте та ініціалізуйте змінні-члени в одному порядку**

Змінні члени завжди ініціалізуються в тому порядку, в якому вони оголошені при визначенні класу; порядок їх згадування у списку ініціалізації конструктора ігнорується.

1. **Уникайте викликів віртуальних функцій у конструкторах та деструкторах**

Усередині конструкторів та деструкторів віртуальні функції втрачають віртуальність.

1. **Проектуйте та пишіть безпечний щодо помилок код**

Переконайтеся, що за будь-яких помилок ваша програма завжди залишається в коректному стані (у цьому полягає базова гарантія). Остерігайтеся помилок, що порушують інваріант (включаючи витоку, але не обмежуючись ними).

1. **Для сповіщення про помилки слід використовувати виключення**

Щоб повідомити про помилки, краще використовувати механізм виключень, а не коди помилок. Застосовувати коди стану слід лише тоді, коли не можна використовувати виключення.

1. **Генеруйте виключення за значенням, перехоплюйте за посиланням**

Генеруйте виключення за значенням (не через покажчик) та перехоплюйте їх як посилання (зазвичай константні). Ця комбінація найкраще відповідає семантиці виключень.

1. **Вважайте за краще виклики алгоритмів циклів, що самостійно розробляються**

Виклик алгоритму замість самостійно розробленого циклу може виявитися більш виразним, легше супроводжуваним, менш схильним до помилок і не менш ефективним.

1. **Уникайте явного вибору типів – використовуйте поліморфізм**

Уникайте явного вибору типу об'єкта для налаштування поведінки. Використовуйте шаблони та віртуальні функції для того, щоб поведінка об'єкта визначалася його типом, а не кодом, що викликає.

1. **Не використовуйте перетворення типів у стилі C**

Старе перетворення типів у стилі С має різну (і часто небезпечну) семантику залежно від контексту, заховану за єдиним синтаксисом. Заміна перетворення типів у стилі З перетвореннями С++ допоможе захиститись від несподіваних помилок.

**Додаткові матеріали**

1. **H. Sutter, A. Alexandrescu “C++ Coding Standards. 101 Rules, Guidelines, and Best Pracitces ”**
2. [**CppCoreGuidelines/CppCoreGuidelines.md at master · isocpp / CppCoreGuidelines (github.com )**](https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md)
3. [**Google C++ Style Guide**](https://google.github.io/styleguide/cppguide.html)