**Анкета рецензента**

Розділ 2. Реалізація

**Бучок** є рецензентом **Zveriok**

**1. Ваша оцінка стилю кодування. Наскільки легко читати і розуміти код.**

Код читати дуже важко, насамперед через дублювання коду.

Закриту частину клас краще розміщувати знизу, оскільки користувачеві класу насамперед цікаві його методи.

Розміщення конструкторів у header файлі доволі незвичне. Зазвичай копіювальний написаний після конвертувальних(-ого), але тут навпаки.

Назва селектора const char\* text() const { return arr; } не є найкращим вибором. Краще було би назвати його c\_str(), набагато зрозуміліше, а також так називається селектор у std::string, тому програмістам, які працювали раніше з std::string буде легше користуватись класом.

using namespace std;

Додавання всього простору імен std до глобального простору, що може спричинити проблеми у майбутньому для користувачів класу.

винесення ітератора за дужки циклу є надлишковим

String::String(const String& str)

{

length = str.length();

unsigned int i;

arr = new char[length];

for (i = 0; i < length; i++)

arr[i] = str[i];

}

Можна було використовувати size\_t замість unsigned, аби покращити читабельність коду.

Нелогічне розміщення команд. Спочатку потрібно присвоїти length, потім виділяючи пам’ять використати length

arr = new char[str.length()];

length = str.length();

strcpy\_s(arr, str.length() + 1, str.c\_str());

**2. Чи наявне дублювання коду?**

Так, наявне. Дублюються оператори для різних типів параметрів.

const String operator+(String&, const String&);

const String operator+(String&, const string&);

const String operator+(String&, const char\*);

String& operator+=(const string&);

String& operator+=(const String&);

String& operator+=(const char\*);

Це ускладнює сприйняття класу і є непотрібним.

Також, можна було не повторювати код у конструкторах, натомість використати делегування конструкторів.

**3. Як реалізовані змішані операції?**

Змішані операції реалізовані через неявні конвертувальні конструктори.

**4. Наскільки ефективна реалізація функцій?**

Конструктори не мають списку ініціалізації, ресурси класу ініціалізуються після входу у конструктор.

У операторі [] є перевірка на те чи ((i < length) && (i >= 0)), i - індекс. Однак друга умова у цій перевірці надлишкова, оскільки параметром є не int, а unsigned int

Не реалізовані оператори порівняння, оскільки їхня сигнатура є сигнатурою утиліт, але вони написані всередині класу.

Перевірка не є правильною. Оператори > < застосовуються для порівняння рядків за лексикографічним порядком.

bool operator<(const String& s1, const String& s2) {

return s1.length() < s2.length();

}

bool operator>(const String& s1, const String& s2) {

return s1.length() > s2.length();

}

Оператор реалізований неправильно, завжди видаватиме false

bool String::operator==(const char\* ch) {

return arr == ch;

}

Нащо використовувати std::string, як буфер?

String& String::operator=(const String& str)

{

length = str.length();

string temp = str.text();

unsigned int i;

arr = new char[length];

for (i = 0; i < length; i++)

arr[i] = temp[i];

return \*this;

}

Загалом втрачено багато оптимізаційних можливостей, зокрема немає семантики переміщень.

**5. Чи передбачено захист від некоректного застосування функцій?**

У операторі + присутній const, щоб запобігати команді a + b = c, де a,b,c - об’єкти класу String.

const String operator+(String&, const String&);

const String operator+(String&, const string&);

const String operator+(String&, const char\*);

В операторі [] присутня перевірка на правильність індексу

if ((i < length) && (i >= 0))

**6. Як реалізована обробка помилок?**

Вона не є реалізованою