# Анкета рецензента

Розділ 1. Специфікація

### 1. Що собою являє даний клас? Яке його призначення?

String є RAII класом облонкою для обробки рядків символів типу char.

Його призначенням є інкапсуляція рядків символів char. Підвищення рівня абстракції полегшує роботу користувача цього класу у порівнянні зі звичайним указником на рядок символів char.

Оскільки, після виходу з області видимості, в якому об’єкт класу String було створено, деструктор класу String гарантовано спрацює, користувачеві класу не потрібно буде звільняти пам’ять самотужки, що призводить як до більш зрозумілого коду, так і до підвищення надійності програми загалом.

Клас String дозволяє користувачеві класу наступні дії:

* Копіювання за допомогою копіювального конструктора та копіювального присвоєння
* Створення об’єкту за допомогою конвертуального конструктором
* Конкатенацію +
* Суміщене присвоєння +=
* Селектор і модифікатор []
* Зведення до типів char\* і string з бібліотеки STL.

Також, передбачена можливість виконання змішаних операцій над рядками власного класу String, класу string з бібліотеки STL та С-рядками.

### 2. Чи існують стандартні рішення цієї ж проблеми, чим вони відрізняються?

Так, існують стандартні рішення, такі як: c-style string, себто просто масив символів типу char, та с++-style string, тобто клас std::string.

C-style

Рядки в стилі С є масивами символів типу char. Існують функції strlen, strcpy, strcat, strcmp, які використовуються для рядків с-style. Масиви C не відстежують власний розмір, але функція strlen повертає довжину рядка без нульового символу, що є не дуже інтуїтивно.

З того факту, що C-style string є масивом витікає, що користуватись цим рішенням потрібно дуже обережно, зокрема стежити за виділенням та звільненням пам’яті самотужки або слідкувати за виходом за межі масиву під час виконання програми.

Багато C-рядків використовуються як масиви фіксованого розміру. Це стосується як літералів, так і масивів, які оголошуються у вигляді char str[n]. Для рядків з динамічним розміром програмісти повинні турбуватися про ручне виділення, зміну розміру та копіювання рядків.

Однак, у такому рішення є і плюси. Для C-рядків пам’ять може виділятись статично, а для String тільки під час виконання програми.

Від String таке рішення відрізняється кардинально, однак від std::string відмінностей вже набагато менше.

C++-style

Мова C++ надає просте, безпечне рішення для роботи з рядками. Клас оболонка string мови C++ string, що є частиною простору імен std, дозволяє абсолютно безпечно маніпулювати рядками.

Клас std::string пропонує:

* Гнучкий обсяг пам'яті
* Отримання довжини рядка за константний час
* Не потрібно турбуватися про ручне управління пам'яттю або зміну розміру рядків
* Проблеми з межами вирішуються без участі користувача
* Інтуїтивне присвоювання з використанням оператора =
* Інтуїтивно зрозуміле порівняння з використанням оператора == замість strcmp
* Інтуїтивно зрозумілі інтерфейси для операцій, зокрема присвоєння, що виглядатиме як =, замість функції strcpy

Однією з основних претензій до std::strings є те, що Ініціалізація std::string з використанням рядкового літералу завжди буде включати копію. Це ж стосується і String.

Хоч і клаc std::string надає набагато більший функціонал для роботи з рядками, у String і std::string схожі в тому, що і один, і другий є класами оболонками над рядками символів char. String має мінімальний набір інструментів для роботи, однак, є безпечним, зручним, інтуїтивним рішенням

### 3. Чи є достатньою документованість інтерфейсу класу, чи є імена методів само документованими?

Документованість класу є злегка недостатньою. Імена методів є самодокументованими, однак є виняток.

Виняток:

* void copyon(char\*, const char\*, size\_t);

### 4. Чи наявні конструктор(можливо декілька) і деструктор?

Так, наявні. Присутні конструктор без параметрів, конструктор з для рядків символів типу char, конструктор для std::string, копіювальний конструктор і деструктор.

### 5. Чи підтримує клас семантику переміщень?

Семантика переміщень не підтримується.

### 6. Наскільки повно інтерфейс відображає призначення класу? Чи всі його атрибути та методи й утиліти коректно та раціонально визначені?

Інтерфейс відображає призначення класу, однак є винятки

Винятки:

* char& operator[](size\_t);

Неможливо використати для сталих об’єктів класу.

* Немає конструктора для одного символа char, лише для рядка символів

На мою думку, майже все визначено коректно, однак є винятки:

Винятки:

* У операторах

bool operator==(const String&, const String&);

bool operator!=(const String&, const String&);

bool operator<(const String&, const String&);

bool operator<=(const String&, const String&);

bool operator>(const String&, const String&);

bool operator>=(const String&, const String&);

Оператори є утилітами, однак визначені всередині класу String.

* У операторах

String& operator=(const string&);

String& operator=(const String&);

String& operator=(const char\*);

String& operator+=(const string&);

String& operator+=(const String&);

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

Оператори визначені, як для класу String, так і рядків символів char і std::string, що створює непотрібний код, позаяк можна оголошувати тільки по одному оператору на кожен оператор використовуючи конвертування типів.

* char& operator[](size\_t);

Застосування оператора[] до константного об’єкту класа String є неможливим

### 7. Чи є поділ класу на закриту і відкриту частину? В який спосіб передбачено доступ до атрибутів?

Так, у класі String є як закрита, так і відкрита частина.

Доступ до атрибута length передбачено методам класу, однак для

char\* arr методу немає.

### 8. Чи є параметри за замовчуванням?

Немає.

### 9. Чи визначені для класу конструктори-конвертори та методи, які можна використати для явного та неявного приведення типів?

Визначені лише конструктори-конвертори для явного приведення типів.

### 10. Чи є в класі friends? Які для цього підстави?

Немає

### 11. Чи коректне застосування const та & в методах та селекторах?

Загалом так, але є винятки.

Винятки:

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

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

Якщо параметр є констант, то ми не зможемо ані звільнити указник, ані присвоїти укразнику nullptr, що змушує користувача класу робити це явно.

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

відсутній const у першого параметру

### 12. Чи наявні утиліти? Які права доступу до атрибутів в них?

Наявні. Вони не мають права доступу безпосередньо до атрибутів.

### 13. Чи є інтерфейс класу мінімальним, які операції не є обов’язковими?

Ні, не є. Є метод для перевірки, чи об’єкт класу String не містить нічого.

### 14. Чи передбачена обробка помилок?

Ні, не передбачено.