

Отчёт по индивидуальному заданию №3

«Класс String»

Стойко Елисей, 5130904/50005

Вариант 11

Содержание

1. Задание	2
2. Требования к реализации	2
3. Структура класса	2
4. Особенности реализации	3
4.1. Оператор ввода (>>)	3
4.2. Оператор < (операторы сравнения)	3
4.3. Конкатенация строк (операторы + и +=)	4
5. Дополнительное задание (вариант 11)	5
6. Тест-план	6
6.1. Конструкторы, методы и функции	6
6.2. Обработка ошибок и сложные случаи	8

1. Задание

Разработайте класс `String`, аналогичный классу `std::string`, с основными операциями для работы со строками.

2. Требования к реализации

Использование строки в стиле C, считая, что символы имеют коды от 0 до 127

Правильное управление памятью (правило трех/пяти)

Использование идиомы `copy-and-swap`

Безопасность исключений

Возможно использование библиотечных функций из `<cctype>`

Запрещается использование функций из `<cstring>`

Тестирование можно выполнить в функции `main`.

3. Структура класса

- `String.h` - объявление класса
- `String.cpp` - определение методов класса и внешних функций
- `Strutils.h` - объявление дополнительных функций для работы со строками
- `Strutils.cpp` - определение дополнительных функций
- `main.cpp` - тестирование класса по тест-плану

4. Особенности реализации

4.1. Оператор ввода (>>)

1. очищаем строку перед записью (метод `clear()`)
2. пропускаем все пробелы до первого *непробельного* символа
3. проверяем состояние потока
4. добавляем к строке по символу из потока до пробела / конца строки
5. символ добавляем в формате `... += { ch, '\0' }` для корректной работы оператора `+=`

```
std::istream& operator>>(std::istream& stream, String& str) {
    str.clear();
    char ch = 0;

    while (stream.get(ch) && std::isspace(ch)) {

    }

    if (!stream) {
        return stream;
    }

    char buf[2] = { ch, '\0' };
    str += buf;

    while (stream.get(ch) && !std::isspace(ch)) {
        char buf[2] = { ch, '\0' };
        str += buf;
    }

    return stream;
}
```

4.2. Оператор < (операторы сравнения)

1. строки сравниваются лексикографически, как в `std::string`
2. проверяем каждый символ до конца минимальной по длине строки
3. если все символы равны, сравниваем длины строк
4. оператор `==` написан аналогично
5. остальные операторы сравнения реализованы через `<` и `==`

```
bool String::operator<(const String& other) const {
    size_t minLen = std::min(length_, other.length_);
    for (size_t i = 0; i < minLen; ++i) {
        if (data_[i] < other.data_[i]) {
            return true;
        }
        if (data_[i] > other.data_[i]) {
            return false;
        }
    }
    return length_ < other.length_;
}
```

4.3. Конкатенация строк (операторы + и +=)

1. оператор + реализован через оператор +=
2. чтобы избежать проблем с копированием символов через `my::strcpy` при самоприсваивании вида `a += a`, используется временная переменная `temp` и рекурсивный вызов функции
3. оператор += для `const char*` реализован через оператор += для `const String&`, путём приведения C-строки к типу `String` через реализованный по условию конструктор

```
String& String::operator+=(const String& other) {
    if (other.length_ == 0 || other.data_ == nullptr) {
        return *this;
    }

    if (this == &other) {
        String temp(other);
        return *this += temp;
    }

    size_t newLen = length_ + other.length_;
    if (newLen + 1 <= capacity_) {
        my::strcpy(data_ + length_, other.data_);
    } else {
        capacity_ = newLen + 1;
        char* newData = new char[capacity_];

        my::strcpy(newData, data_);
        my::strcpy(newData + length_, other.data_);

        delete[] data_;
        data_ = newData;
    }
    length_ = newLen;
    return *this;
}

String String::operator+(const String& other) const {
    String result(*this);
    result += other;
    return result;
}

String& String::operator+=(const char* str) {
    return *this += String(str);
}
```

5. Дополнительное задание (вариант 11)

11. Сформировать новую строку, заменив в исходной строке все прописные латинские буквы на строчные.

```
String String::lower() const {  
    String newString{*this};  
    int delta = 'a' - 'A';  
  
    for (size_t i = 0; i < newString.length_; ++i) {  
        if (newString[i] >= 'A' &&  
            newString[i] <= 'Z' && std::isupper(newString[i])) {  
            newString[i] = newString[i] + delta;  
        }  
    }  
  
    return newString;  
}
```

Метод возвращает новую строку, в которой все символы 'A'..'Z' заменены на соответствующие 'a'..'z'. Остальные символы остаются без изменений.

6. Тест-план

6.1. Конструкторы, методы и функции

№	Тестируемый элемент	Описание теста	Входные данные	Ожидаемый результат
1	<code>String()</code> (констр. по умолчанию)	Пустая строка	<code>String s;</code>	<code>s.length() == 0</code> , <code>s.empty() == true</code>
2	<code>String()</code>	Множественное создание, вычисление длины строки	<code>String s2a;</code> <code>String s2b;</code> <code>String s2c;</code> <code>s2a = "a";</code> <code>s2b = "bb";</code> <code>s2c = "ccc";</code>	1 2 3
3	<code>String(const char*)</code>	Пустой литерал	<code>String s("");</code>	<code>length() == 0</code> , <code>data_[0] == '\0'</code>
4	<code>String(const char*)</code>	Обычная строка	<code>"Hello"</code>	<code>length() == 5</code> , последние символы совпадают
5	<code>String(const char*)</code>	Длинная строка	Строка длиной 1000+ символов	Корректная работа, нет переполнений
6	<code>String(const String&)</code>	Копия непустой строки	<code>String a("abc");</code> <code>String b(a);</code>	<code>b == a</code> , но <code>b.data_ != a.data_</code>
7	<code>String(const String&)</code>	Копия пустой строки	<code>String a;</code> <code>String b(a);</code>	<code>b.empty() == true</code>
8	<code>String(String&&)</code>	Перемещение временного объекта	<code>String b(String("abc"));</code>	<code>b == "abc"</code> , у источника <code>length_ == 0</code> , <code>data_</code> в безопасном состоянии
9	<code>String(String&&)</code>	Перемещение непустой строки	<code>String a("abc");</code> <code>String b(std::move(a));</code>	<code>b == "abc"</code> , <code>a</code> в валидном «пустом» состоянии
10	<code>length()</code>	Пустая строка	<code>String s;</code>	<code>length() == 0</code>
11	<code>length()</code>	Непустая строка	<code>"Test"</code>	<code>length() == 4</code>
12	<code>empty()</code>	Пустая строка	<code>String s;</code> или <code>String s("");</code>	<code>empty() == true</code>
13	<code>empty()</code>	Непустая строка	<code>"a"</code>	<code>empty() == false</code>
14	<code>operator=(const String&)</code>	Присваивание между разными объектами	<code>a = "abc";</code> <code>b = "x";</code> <code>b = a;</code>	<code>b == "abc"</code> , старый буфер <code>b</code> освобождён
15	<code>operator=(const String&)</code>	Самоприсваивание	<code>a = "abc";</code> <code>a = a;</code>	Состояние не меняется, нет падения

16	<code>operator=(String&&)</code>	Присваивание перемещением	<code>a = "abc"; b = std::move(a);</code>	<code>b == "abc",</code> a в «пустом» состоянии
17	<code>operator=(const char*)</code>	Присваивание пустой C-строки	<code>s = "hello"; s = "";</code>	<code>s.empty() == true</code>
18	<code>operator=(const char*)</code>	Длинная строка	<code>s = "a"; s = длинная_строка</code>	Буфер перераспределён, строка корректна
19	<code>operator[](size_t)</code>	Первый символ	<code>String s("abc"); s[0] = 'z';</code>	Строка "zbc"
20	<code>operator[](size_t)</code>	Последний символ	<code>s[length() - 1]</code>	Меняется только последний символ
21	<code>operator[](size_t) const</code>	Чтение	<code>const String s("abc"); char c = s[1];</code>	<code>c == 'b'</code>
22	<code>operator+</code>	Оба непустые	<code>"ab" + "cd"</code>	"abcd"
23	<code>operator+</code>	Левая пустая	<code>String() + "abc"</code>	"abc"
24	<code>operator+</code>	Правая пустая	<code>"abc" + String()</code>	"abc"
25	<code>operator+=(const String&)</code>	Добавление к непустой	<code>"ab" += "cd"</code>	"abcd"
26	<code>operator+=(const String&)</code>	Добавление пустой строки	<code>"ab" += ""</code>	"ab"
27	<code>operator+=(const char*)</code>	Добавление C-строки	<code>"ab" += "cde"</code>	"abcde"
28	<code>clear()</code>	Очищение непустой строки	<code>s = "abc"; s.clear();</code>	<code>s.empty() == true,</code> <code>data_[0] == '\0'</code>
29	<code>clear()</code>	Множественный вызов	<code>s = "abc"; s.clear(); s.clear();</code>	Корректная работа, без падения
30	<code>operator==</code>	Равные строки	<code>"abc" == "abc"</code>	true
31	<code>operator==</code>	Разные строки	<code>"abc" == "abd"</code>	false
32	<code>operator!=</code>	Разные строки	<code>"abc" != "abd"</code>	true
33	<code>operator<</code>	Лексикографическое сравнение	<code>"abc" < "abd"</code>	true
34	<code>operator<</code>	Префикс	<code>"ab" < "abc"</code>	true
35	<code>operator<=, >=</code>	Равные строки	<code>"abc" <= "abc"</code>	true
36	<code>operator>></code>	Ввод одного слова	ввод: hello↵	Строка "hello"
37	<code>operator>></code>	Ввод до пробела	ввод: hello world↵	Строка "hello", остаток во входном потоке
38	<code>operator>></code>	Пустой ввод	нажатие Enter сразу	Строка пустая или предыдущее значение

39	<code>operator<<</code>	Вывод непустой	<code>String("abc")</code>	На экране "abc" без лишних символов
40	<code>operator<<</code>	Вывод пустой	<code>String()</code>	Пустая строка, только перевод строки (если добавить)
41	<code>lower()</code>	Смешанный регистр	<code>"AbC1!"</code>	<code>"abc1!"</code>
42	<code>lower()</code>	Нет латинских букв	<code>"123_+ -"</code>	Строка без изменений
43	<code>lower()</code>	Уже нижний регистр	<code>"abc"</code>	Строка без изменений
44	<code>my::strlen</code>	Пустая C-строка	<code>""</code>	0
45	<code>my::strlen</code>	Обычная строка	<code>"abcd"</code>	4
46	<code>my::strcpy</code>	Копирование обычной строки	<code>dest</code> достаточно велик, <code>src "abc"</code>	В <code>dest</code> <code>"abc\0"</code>
47	<code>my::strcpy</code>	Копирование пустой строки	<code>src ""</code>	В <code>dest</code> сразу <code>'\0'</code>

6.2. Обработка ошибок и сложные случаи

№	Тестируемый элемент	Описание теста	Входные данные	Ожидаемый результат
1	<code>operator[](size_t)</code>	Индекс > <code>length()</code> или Индекс < 0	<code>String s("abc");</code> <code>s[3] = 'd';</code>	ERROR: string index out of bounds
2	<code>operator[](size_t) const</code>	Индекс > <code>length()</code> или Индекс < 0	<code>const String s("abc");</code> <code>s[3];</code>	ERROR: string index out of bounds
3	<code>operator+=(const String&)</code>	<u>Самоприсваивание при сложении вида:</u> <code>a += a</code>	<code>String s1("abc");</code> <code>s1 += s1;</code>	Строка "abcabc"
4	<code>my::strcpy</code>	Самокопирование (<code>dest == src</code>)	Особый случай	UB