Строки – 1

Denis Bakin

std::string – особый вектор

- `std::string` — контейнер символов, похожий на `std::vector<char>`
- Многие методы вектора доступны: `size()`, `push_back()`, `pop_back()`, `resize()`
- Дополнительные методы для работы с текстом

Конкатенация строк

```cpp
#include <iostream>
#include <string>

int main() {
    std::string s = "Some string";

    s += ' ';           // добавляем символ (аналог push_back)
    s += "functions";   // добавляем строку в конец

    std::cout << s << "\n";  // Some string functions
}
```

- Оператор += работает и с `char`, и с `const char*`

Выделение подстроки: substr

```
std::string s = "Some string functions";

// substr(pos, len) -- подстрока длины len с позиции pos
std::string sub1 = s.substr(5, 6);   // "string"

// substr(pos) -- подстрока с позиции pos до конца
std::string sub2 = s.substr(12);     // "functions"
```

- Первый аргумент – начальная позиция (с 0)
- Второй аргумент – длина (необязательный)

Поиск в строке: find

```cpp
std::string s = "Some string functions";

size_t pos1 = s.find(' ');          // 4 (первый пробел)
size_t pos2 = s.find(' ', pos1 + 1); // 11 (следующий пробел)
size_t pos3 = s.find("str");        // 5 (позиция подстроки)
size_t pos4 = s.find("#");          // std::string::npos
```

- Возвращает позицию первого вхождения
- `std::string::npos` — если не найдено

Проверка результата find

```
std::string s = "Hello, world!";

size_t pos = s.find("world");

if (pos != std::string::npos) {
    std::cout << "Найдено на позиции " << pos << "\n";
} else {
    std::cout << "Не найдено\n";
}
```

- Всегда проверяйте результат на npos!

Вставка подстроки: insert

```
std::string s = "Some string functions";

// insert(pos, str) -- вставляет str перед позицией pos
s.insert(5, "std::");

std::cout << s << "\n";  // Some std::string functions
```

- Первый аргумент – позиция вставки
- Второй аргумент – вставляемая строка

Замена подстроки: replace

```
std::string s = "Some std::string functions";

// replace(pos, len, str) -- заменяет len символов с позиции pos
s.replace(0, 4, "Special");

std::cout << s << "\n";  // Special std::string functions
```

- Первый аргумент – начальная позиция
- Второй аргумент – длина заменяемой части
- Третий аргумент – новая подстрока

Удаление подстроки: erase

```
std::string s = "Special std::string functions";

// erase(pos, len) -- удаляет len символов с позиции pos
s.erase(8, 5);  // удаляем "std::"

std::cout << s << "\n";  // Special string functions
```

- Аналогичный синтаксис: позиция + длина

Проверка префикса и суффикса (C++20)

```cpp
#include <iostream>
#include <string>

int main() {
    std::string phrase;
    std::getline(std::cin, phrase);

    if (phrase.starts_with("hello")) {
        std::cout << "Greeting\n";
    }

    if (phrase.ends_with("bye")) {
        std::cout << "Farewell\n";
    }
}
```

- `starts_with` – проверка начала строки
- `ends_with` – проверка конца строки

Бесконечный ввод: концепция

- В Unix “everything is a file”
- Потоки ввода/вывода – как бесконечные файлы
- EOF (End Of File) – специальный символ конца
- В консоли: `Ctrl+D` (Unix)

Чтение до конца ввода

```
int a = 0, b = 0;

// Цикл продолжается, пока чтение успешно
while (std::cin >> a >> b) {
    // обработка a и b
}
```

- Оператор >> возвращает `true` при успешном чтении
- Цикл завершается при EOF или ошибке

Чтение строки целиком: getline

```cpp
#include <iostream>
#include <string>

int main() {
    std::string line;

    // Читает до символа '\n'
    std::getline(std::cin, line);

    std::cout << "Вы ввели: " << line << "\n";
}
```

- `std::getline` читает до конца строки
- Включает пробелы (в отличие от >>)

Разбор строки: stringstream

```cpp
#include <iostream>
#include <sstream>
#include <vector>

int main() {
    std::string line;
    std::getline(std::cin, line);  // "1 2 3 4 5"

    std::stringstream ss(line);
    std::vector<int> numbers;
    int num;

    while (ss >> num) {
        numbers.push_back(num);
    }
    // numbers = {1, 2, 3, 4, 5}
}
```

stringstream – поток из строки

- `std::stringstream` – поток, работающий со строкой
- Поддерживает оператор >> как `std::cin`
- Удобен для разбора строки на части

```
std::stringstream ss("42 3.14 hello");
int i;
double d;
std::string s;

ss >> i >> d >> s;
// i = 42, d = 3.14, s = "hello"
```

Полный пример: чтение по строкам

```cpp
#include <iostream>
#include <vector>
#include <sstream>

int main() {
    std::vector<int> numbers;
    std::string line;

    while (std::getline(std::cin, line)) {
        std::stringstream ss(line);
        int num;
        while (ss >> num) {
            numbers.push_back(num);
        }
    }
}
```

Сводка методов std::string

Метод	Описание
`+=`	Добавление символа/строки в конец
`substr(pos, len)`	Выделение подстроки
`find(str, pos)`	Поиск подстроки
`insert(pos, str)`	Вставка подстроки
`replace(pos, len, str)`	Замена подстроки
`erase(pos, len)`	Удаление подстроки

Битовые операции: обзор

- Работают с отдельными битами числа
- Основные операции в C++:

Оператор	Название	Пример
`&`	AND (И)	`5 & 3 = 1`
`|`	OR (ИЛИ)	`5 | 3 = 7`
`^`	XOR (искл. ИЛИ)	`5 ^ 3 = 6`
`~`	NOT (НЕ)	`~5 = -6`
`<<`	Сдвиг влево	`5 << 1 = 10`
`>>`	Сдвиг вправо	`5 >> 1 = 2`

Битовые операции: примеры

```
uint8_t a = 0b00000101;  // 5
uint8_t b = 0b00000011;  // 3

uint8_t and_result = a & b;  // 0b00000001 = 1
uint8_t or_result  = a | b;  // 0b00000111 = 7
uint8_t xor_result = a ^ b;  // 0b00000110 = 6
```

- `&` – бит равен 1, если оба бита равны 1
- `|` – бит равен 1, если хотя бы один бит равен 1
- `^` – бит равен 1, если биты различаются

Битовые сдвиги

```
uint8_t x = 0b00000101;  // 5

uint8_t left  = x << 2;  // 0b00010100 = 20
uint8_t right = x >> 1;  // 0b00000010 = 2
```

- `<< n` – умножение на 2^n
- `>> n` – деление на 2^n (целочисленное)

Маскирование битов

```
uint8_t byte = 0b11010110;

// Извлечь младшие 4 бита
uint8_t low = byte & 0x0F;       // 0b00000110 = 6

// Извлечь старшие 4 бита
uint8_t high = (byte >> 4) & 0x0F;  // 0b00001101 = 13

// Установить бит 3
byte = byte | (1 << 3);          // 0b11011110

// Сбросить бит 2
byte = byte & ~(1 << 2);         // 0b11011010
```

Кодировка UTF-8

- UTF-8 – кодировка переменной длины (1-4 байта)
- Совместима с ASCII (первые 128 символов)
- Самая распространённая кодировка в интернете

UTF-8: структура байтов

Диапазон кодов	Байт 1	Байт 2	Байт 3	Байт 4
U+0000..U+007F	0xxxxxxx	—	—	—
U+0080..U+07FF	110yyyyy	10xxxxxx	—	—
U+0800..U+FFFF	1110zzzz	10yyyyyy	10xxxxxx	—
U+10000..U+10FFFF	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

- Первый байт определяет длину последовательности
- Продолжающие байты начинаются с 10

UTF-8: определение длины

```
int getUtf8Length(uint8_t firstByte) {
    if ((firstByte & 0x80) == 0x00) return 1;  // 0xxxxxxx
    if ((firstByte & 0xE0) == 0xC0) return 2;  // 110xxxxx
    if ((firstByte & 0xF0) == 0xE0) return 3;  // 1110xxxx
    if ((firstByte & 0xF8) == 0xF0) return 4;  // 11110xxx
    return -1;  // ошибка
}
```

- Проверяем старшие биты первого байта
- Маска выделяет нужные биты для сравнения

UTF-8: декодирование (3 байта)

```
// Декодирование 3-байтовой последовательности
// 1110zzzz 10yyyyyy 10xxxxxx → zzzzyyyy yyxxxxxx
uint8_t b1 = 0xE4, b2 = 0xB8, b3 = 0xAD;  // китайский иероглиф 中

uint32_t codepoint = ((b1 & 0x0F) << 12)  // zzzz
                   | ((b2 & 0x3F) << 6)    // yyyyyy
                   | (b3 & 0x3F);          // xxxxxx
// codepoint = 0x4E2D (U+4E2D = 中)
```

UTF-8: кодирование

```
void encodeUtf8(uint32_t codepoint, std::string& out) {
    if (codepoint <= 0x7F) {
        out += static_cast<char>(codepoint);
    } else if (codepoint <= 0x7FF) {
        out += static_cast<char>(0xC0 | (codepoint >> 6));
        out += static_cast<char>(0x80 | (codepoint & 0x3F));
    } else if (codepoint <= ...) {
        ...
    }
}
```

UTF-8 и std::string

- `std::string` хранит байты, не символы Unicode
- `s.size()` возвращает число байтов, не символов
- Для корректной работы с Unicode нужна специальная обработка

```
std::string s = "Привет";  // UTF-8
std::cout << s.size() << "\n";  // 12 (не 6!)
// Каждая кириллическая буква = 2 байта
```