Лекція 4

Колекції: Vec<T>

Динамічні дані у Rust

push • pop • iter • slice • capacity

🤖 Приклади: координати БПЛА, черга команд, рій агентів

Частина 1: Основи та базові операції

О. С. Бичков • 2025

План лекції (Частина 1)

1. Навіщо потрібен Vec?
2. Що таке Vec<T>?
3. Stack vs Heap
4. Створення векторів
5. push / pop
6. insert / remove
7. Доступ до елементів
8. len, capacity, is_empty
9. Реалокація
10. reserve, shrink, clear
11. Ownership
12. 🤖 Рій БПЛА як Vec
13. 🤖 Черга команд
14. 🤖 Маршрут польоту

Частина 2: Slices, ітерація, функціональний стиль, MAC приклади

Навіщо потрібен Vec?

Проблема: Масиви [T; N] мають фіксований розмір на етапі компіляції

Але в реальних програмах кількість елементів часто невідома заздалегідь!

🚁 БПЛА
Скільки точок маршруту?
Скільки цілей?

📋 Веб
Скільки користувачів?
Скільки товарів?

🎮 Гра
Скільки ворогів?
Скільки предметів?

Рішення: Vec<T> — вектор, що росте динамічно під час виконання!

```rust
// Вектор чисел
let nums: Vec<i32> = vec![1, 2, 3];

// Вектор рядків
let names: Vec<String> = vec![];
```

Структура в пам'яті:

```
Stack:  [ptr | len: 3 | cap: 4]
          ↓
Heap:   [1] [2] [3] [?]
```

ptr — вказівник на heap
len — кількість елементів
cap — виділено місця

Stack vs Heap: чому це важливо?

```
// На стеку
let arr: [i32; 3] = [1, 2, 3];
let x: i32 = 42;
```

Heap (купа)

✓ Динамічний розмір
✓ Великі обсяги даних
~ Повільніший доступ
~ Потребує управління

Rust: Vec автоматично звільняє heap-пам'ять при виході зі scope (RAII)

```
// На heap
let v: Vec<i32> = vec![1, 2, 3];
let s = String::from("hi");
```

Способи створення Vec

```
let mut v = Vec::with_capacity(100);
// len=0, capacity=100
```

4. Vec::from()

💡 with_capacity — якщо знаєте приблизну кількість елементів

```
let v = Vec::from([1, 2, 3]);
let bytes = Vec::from("hello");
```

```
// Структура позиції дрона
struct Position { x: f64, y: f64, altitude: f64 }

// Структура дрона
struct Drone {
    id: u32,
    position: Position,
    battery: u8,
}

// Рій дронів — вектор!
let mut swarm: Vec<Drone> = Vec::with_capacity(50);

// Маршрут польоту — вектор точок
let flight_path: Vec<Position> = vec![
    Position { x: 0.0, y: 0.0, altitude: 100.0 },
    Position { x: 100.0, y: 50.0, altitude: 150.0 },
];
```

Чому Vec? Кількість дронів, точок маршруту, виявлених цілей — все це динамічне!

Додавання та видалення: push / pop

```
let mut v = vec![1, 2];
v.push(3);
v.push(4);
// v = [1, 2, 3, 4]
```

⏱️ O(1) амортизована
⚠️ Потребує mut

pop() — видалити з кінця

⏱️ O(1) завжди
↩️ Повертає Option<T>

Стек на Vec: push/pop з кінця — ідеально для LIFO!

```
let mut v = vec![1, 2, 3];
let last = v.pop();
// last = Some(3)
```

Вставка та видалення за індексом

```
let mut v = vec![1, 2, 4];
v.insert(2, 3);
// v = [1, 2, 3, 4]
```

⚠️ O(n) — зсуває елементи

remove(index)

⚠️ O(n) — зсуває елементи

💡 swap_remove(index) — O(1)!
Якщо порядок не важливий — міняє з останнім і видаляє

```
let mut v = vec!["a", "b", "c"];
let r = v.remove(1);
// r="b", v=["a", "c"]
```

Доступ до елементів

```
let v = vec![10,20,30];
let first = v[0]; // 10
// v[10] → panic!
```

💥 Паніка за межами

get() — безпечно

✓ Option<&T>

first() / last()

✓ Зручно для черг

Правило: get() коли індекс може бути невалідним, [] коли впевнені

```
v.get(0)  // Some(&10)
v.get(10) // None
```

```rust
// first_mut() / last_mut()
let mut v = vec!["a", "b", "c"];
if let Some(last) = v.last_mut() {
    *last = "z";
}
// v = ["a", "b", "z"]
```

```rust
if let Some(elem) = v.get_mut(0) {
    *elem = 100;
}
```

Розмір та ємність

```
let v = vec![1, 2, 3];
v.len() // 3
```

Кількість елементів

is_empty()

len == 0

capacity()

Виділено пам'яті

len — скільки використовується | capacity — скільки виділено без реалокації

```
let v: Vec<i32> = vec![];
v.is_empty() // true
```

```
let mut v = Vec::new();
for i in 0..10 {
    v.push(i);
    println!("len: {}, cap: {}", v.len(), v.capacity());
}
// len: 1, cap: 4
// len: 2, cap: 4
// len: 5, cap: 8  ← реалокація!
// len: 9, cap: 16 ← реалокація!
```

💡 Vec::with_capacity(n) — уникнути реалокацій, якщо знаєте приблизний розмір

```rust
// truncate(n) — обрізати до n елементів
let mut v = vec![1, 2, 3, 4, 5];
v.truncate(2); // v = [1, 2]
```

```rust
v.shrink_to_fit();
// cap ~= len
```

Ownership: Vec володіє елементами

```
let v = vec![String::from("a")];
let s = v[0];
// ❌ cannot move out
```

✓ Clone

✓ Посилання

Copy типи: Для i32, f64, bool — v[0] копіює автоматично

```
// ✓ Копія
```

```
// Команди для дрона
enum DroneCommand {
    TakeOff, Land, FlyTo(Position), Scan, Return,
}

// Черга команд — Vec!
let mut command_queue: Vec<DroneCommand> = vec![];

// Додавання команд
command_queue.push(DroneCommand::TakeOff);
command_queue.push(DroneCommand::FlyTo(target));
command_queue.push(DroneCommand::Scan);

// Виконання (FIFO — з початку)
while !command_queue.is_empty() {
    let cmd = command_queue.remove(0);
    execute(cmd);
}
```

💡 Для справжньої черги краще VecDeque (O(1) з обох кінців)

```
struct Swarm {
    drones: Vec<Drone>,
    commander_id: Option<u32>,
}

impl Swarm {
    // Додати дрон до рою
    fn add_drone(&mut self, drone: Drone) {
        self.drones.push(drone);
    }

    // Знайти дрон за ID
    fn find_drone(&self, id: u32) -> Option<&Drone> {
        self.drones.iter().find(|d| d.id == id)
    }

    // Кількість активних
    fn active_count(&self) -> usize {
        self.drones.iter().filter(|d| d.is_active()).count()
    }
}
```

```
struct FlightPath {
    waypoints: Vec<Position>,
    current_index: usize,
}

impl FlightPath {
    fn new() -> Self {
        FlightPath { waypoints: vec![], current_index: 0 }
    }

    fn add_waypoint(&mut self, pos: Position) {
        self.waypoints.push(pos);
    }

    fn next_waypoint(&mut self) -> Option<&Position> {
        if self.current_index < self.waypoints.len() {
            let wp = &self.waypoints[self.current_index];
            self.current_index += 1;
            Some(wp)
        } else {
            None
        }
    }
}
```

Основні методи Vec (1/2)

Метод	Опис	Складність
push(elem)	Додати в кінець	O(1)*
pop()	Видалити з кінця → Option<T>	O(1)
insert(i, elem)	Вставити на позицію i	O(n)
remove(i)	Видалити з позиції i → T	O(n)
swap_remove(i)	Видалити, замінивши останнім → T	O(1)
get(i)	Отримати посилання → Option<&T>	O(1)
first() / last()	Перший / останній → Option<&T>	O(1)

* амортизована складність

Основні методи Vec (2/2)

Метод	Опис	Складність
len()	Кількість елементів	O(1)
is_empty()	Чи порожній?	O(1)
capacity()	Виділено пам'яті	O(1)
clear()	Видалити всі елементи	O(n)
reserve(n)	Гарантувати місце для n	O(n)
shrink_to_fit()	Зменшити capacity	O(n)
contains(&elem)	Чи містить елемент?	O(n)
sort()	Відсортувати	O(n log n)

Підсумок: Частина 1

Vec<T> — динамічний масив на heap
• Росте автоматично (реалокація × 2)
• Володіє елементами (Ownership)
• RAII — автоматичне звільнення

Основні операції:
• push/pop — O(1) з кінця
• insert/remove — O(n) за індексом
• get/[] — доступ до елементів

🤖 MAC застосування:
• Рій дронів як Vec<Drone>
• Черга команд, маршрут польоту

→ Частина 2: Slices, ітерація, функціональний стиль

Лекція 4 (продовження)

Колекції: Vec<T>

Slices, ітерація, функціональний стиль

iter • map • filter • collect • enumerate

🤖 Приклади: фільтрація дронів, обробка цілей, статистика рою

Частина 2: Просунуті операції

О. С. Бичков • 2025

План лекції (Частина 2)

1. Slices (зрізи) &[T]
2. Методи slice
3. Ітерація: iter, iter_mut, into_iter
4. enumerate
5. map та filter
6. fold та reduce
7. collect
8. Сортування та пошук
9. retain, dedup
10. extend, append, drain
11. 🤖 Фільтрація дронів
12. 🤖 Обробка цілей
13. 🤖 Статистика рою
14. Практичні поради

Slice — &[T]

```rust
let v = vec![0, 1, 2, 3, 4, 5];

// Весь вектор як slice
let all: &[i32] = &v[..];

// Від 1 до 4 (не включно)
let mid: &[i32] = &v[1..4];    // [1, 2, 3]

// Від початку до 3
let start = &v[..3];           // [0, 1, 2]

// Від 3 до кінця
let end = &v[3..];             // [3, 4, 5]

// Включно (..=)
let incl = &v[1..=3];          // [1, 2, 3]
```

&[T] — іммутабельний slice | &mut [T] — мутабельний slice

Корисні методи slice

```
v.contains(&3) // true
```

starts_with / ends_with

reverse / sort

```
v.starts_with(&[0, 1])
```

Три види ітерації

```
let v = vec![1, 2, 3];
for x in v.iter() {
    println!("{}", x);
}
// v ще доступний!
```

Позичання

iter_mut() — &mut T

Мутабельне позичання

into_iter() — T

Move (споживання)

Скорочення: for x in &v = iter() | for x in &mut v = iter_mut() | for x in v = into_iter()

```
let mut v = vec![1, 2, 3];
for x in v.iter_mut() {
    *x *= 2;
}
// v = [2, 4, 6]
```

```rust
let colors = vec!["red", "green", "blue"];

// enumerate() повертає (індекс, значення)
for (index, color) in colors.iter().enumerate() {
    println!("{}. {}", index + 1, color);
}
// Вивід:
// 1. red
// 2. green
// 3. blue

// Зміна з індексом
for (i, x) in v.iter_mut().enumerate() {
    *x = i as i32;
}

// Пошук індексу
let pos = v.iter().position(|&x| x == 42); // Option<usize>
```

```rust
let v = vec![1, 2, 3, 4, 5];

// Подвоїти кожен елемент
let doubled: Vec<i32> = v.iter()
    .map(|x| x * 2)
    .collect();
// doubled = [2, 4, 6, 8, 10]

// Перетворити в рядки
let strings: Vec<String> = v.iter()
    .map(|x| x.to_string())
    .collect();
// strings = ["1", "2", "3", "4", "5"]

// map не модифікує оригінал — створює новий ітератор
```

map — lazy! Потребує collect() або for для виконання

```rust
let v = vec![1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Тільки парні
let even: Vec<&i32> = v.iter()
    .filter(|x| *x % 2 == 0)
    .collect();
// even = [&2, &4, &6, &8, &10]

// filter + map (ланцюжок)
let result: Vec<i32> = v.iter()
    .filter(|x| *x % 2 == 0)   // парні
    .map(|x| x * 10)            // × 10
    .collect();
// result = [20, 40, 60, 80, 100]
```

filter_map — комбінує filter + map, пропускає None

```rust
// Скорочення: sum(), product(), max(), min()
let sum: i32 = v.iter().sum();
let max = v.iter().max();
```

```rust
let v = vec![1, 2, 3, 4];
let sum = v.iter()
    .copied()
```

```rust
let v = vec![1, 2, 3, 4, 5];

// В новий Vec
let doubled: Vec<i32> = v.iter().map(|x| x * 2).collect();

// В HashSet (унікальні)
use std::collections::HashSet;
let unique: HashSet<i32> = v.into_iter().collect();

// В String
let chars = vec!['H', 'e', 'l', 'l', 'o'];
let s: String = chars.into_iter().collect();
// s = "Hello"

// Turbofish ::<> — явно вказати тип
let v2 = (0..5).collect::<Vec<i32>>();
```

collect() потребує знати цільовий тип — через анотацію або turbofish

```
// Інші методи пошуку
v.iter().find(|x| **x > 3);           // Option<&T>
v.iter().position(|x| *x == 5);       // Option<usize>
v.iter().any(|x| *x > 10);            // bool
v.iter().all(|x| *x > 0);             // bool
```

```
let v = vec![1, 3, 5, 7, 9];
match v.binary_search(&5) {
    Ok(idx) => println!("Found: {}", idx),
```

```rust
// Для всіх дублікатів — спочатку сортуємо
let mut v = vec![1, 3, 2, 1, 3];
v.sort();
v.dedup();
// v = [1, 2, 3]
```

```rust
let mut v = vec![1, 1, 2, 2, 3];
v.dedup();
```

```
// split_off — відрізати хвіст
let mut v = vec![1, 2, 3, 4, 5];
let tail = v.split_off(3);  // tail = [4, 5], v = [1, 2, 3]
```

```
let mut v1 = vec![1, 2];
let mut v2 = vec![3, 4];
v1.append(&mut v2);
// v2 = []
```

```rust
impl Swarm {
    /// Дрони з низьким зарядом (< 20%)
    fn low_battery_drones(&self) -> Vec<&Drone> {
        self.drones.iter()
            .filter(|d| d.battery < 20)
            .collect()
    }

    /// Активні дрони певної ролі
    fn drones_by_role(&self, role: DroneRole) -> Vec<&Drone> {
        self.drones.iter()
            .filter(|d| d.role == role && d.is_active())
            .collect()
    }

    /// Дрони в радіусі від точки
    fn drones_in_radius(&self, center: Position, radius: f64) -> Vec<&Drone> {
        self.drones.iter()
            .filter(|d| d.position.distance_to(&center) <= radius)
            .collect()
    }
}
```

```
struct Target {
    id: u32,
    position: Position,
    threat_level: u8,
    confidence: f32,
}

impl Swarm {
    /// Високопріоритетні цілі (відсортовані за загрозою)
    fn priority_targets(&self, targets: &[Target]) -> Vec<&Target> {
        let mut high_priority: Vec<_> = targets.iter()
            .filter(|t| t.confidence > 0.8 && t.threat_level > 5)
            .collect();

        high_priority.sort_by(|a, b| b.threat_level.cmp(&a.threat_level));
        high_priority
    }

    /// Найближча ціль до дрона
    fn nearest_target(&self, drone: &Drone, targets: &[Target]) -> Option<&Target> {
        targets.iter()
            .min_by_key(|t| drone.position.distance_to(&t.position) as i64)
    }
}
```

```rust
impl Swarm {
    /// Середній заряд батареї
    fn average_battery(&self) -> f32 {
        if self.drones.is_empty() { return 0.0; }
        let total: u32 = self.drones.iter()
            .map(|d| d.battery as u32)
            .sum();
        total as f32 / self.drones.len() as f32
    }

    /// Кількість дронів за ролями
    fn count_by_role(&self) -> HashMap<DroneRole, usize> {
        self.drones.iter()
            .map(|d| d.role)
            .fold(HashMap::new(), |mut acc, role| {
                *acc.entry(role).or_insert(0) += 1;
                acc
            })
    }

    /// Мінімальний заряд
    fn min_battery(&self) -> Option<u8> {
        self.drones.iter().map(|d| d.battery).min()
    }
}
```

```rust
// Розподіл дронів по секторах
fn assign_sectors(drones: &mut Vec<Drone>, sectors: &[Sector]) {
    let drones_per_sector = drones.len() / sectors.len();

    for (i, chunk) in drones.chunks_mut(drones_per_sector).enumerate() {
        if i < sectors.len() {
            for drone in chunk {
                drone.assigned_sector = Some(sectors[i].clone());
            }
        }
    }
}

// Знайти найближчого сусіда для кожного дрона
fn find_nearest_neighbors(drones: &[Drone]) -> Vec<Option<u32>> {
    drones.iter().map(|d| {
        drones.iter()
            .filter(|other| other.id != d.id)
            .min_by_key(|other| d.position.distance_to(&other.position) as i64)
            .map(|other| other.id)
    }).collect()
}
```

Типові помилки з Vec

```
let v = vec![String::from("a")];
let s = v[0]; // ❌ cannot move
```

❌ Забули collect()

```
let doubled = v.iter().map(|x| x*2);
// doubled - ітератор, не Vec!
```

Практичні поради

🎯 Оптимізація:
• with_capacity() — якщо знаєте розмір заздалегідь
• shrink_to_fit() — звільнити зайву пам'ять
• swap_remove() — O(1) якщо порядок не важливий

🔧 Вибір методу:
• get() замість [] — якщо індекс може бути невалідним
• iter() для читання, iter_mut() для зміни
• into_iter() — якщо вектор більше не потрібен

🤖 Для MAC:
• Vec<Agent> — основна структура для рою
• filter + collect — вибірка агентів за критерієм
• sort_by_key — пріоритизація задач
• chunks — розподіл по групах

Vec vs інші колекції

Колекція	Застосування	Особливості
Vec<T>	Динамічний масив	Швидкий доступ за індексом
VecDeque<T>	Черга з двох кінців	O(1) push/pop з обох боків
LinkedList<T>	Зв'язаний список	O(1) вставка в середину
HashMap<K,V>	Ключ-значення	O(1) пошук за ключем
HashSet<T>	Унікальні значення	O(1) перевірка наявності
BTreeMap<K,V>	Впорядкований map	Відсортовані ключі

Vec — найчастіше використовувана колекція. Інші — для специфічних задач.

Підсумок: Частина 2

Slices &[T]:
• Посилання на частину без копіювання
• Range синтаксис: [1..4], [..3], [3..]

Ітерація:
• iter() / iter_mut() / into_iter()
• enumerate для індексу

Функціональний стиль:
• map — перетворити
• filter — відфільтрувати
• fold/reduce — агрегувати
• collect — зібрати

🤖 MAC:
• Фільтрація дронів, обробка цілей, статистика рою

→ Наступна лекція: HashMap та просторові структури

Завдання для самостійної роботи

1. Базове: Створіть Vec<i32>, заповніть числами 1-100, відфільтруйте парні, подвойте їх та виведіть суму.

2. Рій: Реалізуйте структуру Swarm з методами:
   - add_drone(), remove_drone(id)
   - find_nearest(position) — найближчий дрон
   - average_battery() — середній заряд

3. Маршрут: Створіть FlightPath з Vec<Position>:
   - add_waypoint(), remove_waypoint(index)
   - total_distance() — загальна довжина
   - optimize() — видалити близькі точки

4. Цілі: Реалізуйте обробку Vec<Target>:
   - sort_by_priority() — за рівнем загрози
   - filter_confident(threshold) — впевнені цілі
   - nearest_to(drone) — найближча до дрона

Дякую за увагу!

Колекції: Vec<T>

Питання?