Intergerlist

```rust
use std::{io::{self, prelude::*}, vec, collections::VecDeque, panic::resume_unwind};

Debug | ▶ Run | Debug
fn main() {
    let mut input: String = String::new();
    io::stdin().read_to_string(buf: &mut input).expect(msg: "Lecture Stdin");
    let mut lignes: Lines = input.lines();

    let word1: &str = lignes.next().expect(msg: "ddsqdsq");
    let line1: usize = word1.parse::<usize>().expect(msg: "Salut");

    for _i: usize  in 0..line1{
        let l1: Chars = lignes.next().expect(msg: "sasas").chars();
        let _l2: usize = lignes.next().expect(msg: "sasas").parse::<usize>().expect(msg: "Tasqdsq");
        let line3: &str = lignes.next().expect(msg: "Ssddq");
        let first_last_off: &str = &line3[1..line3.len() - 1];

        let l3: Split<&str> = first_last_off.split(",");
        let mut vec: VecDeque<usize> = VecDeque::new();

        for element: &str in l3{
            if element != "" {
                vec.push_back(element.parse::<usize>().expect(msg: "sqdsqd"));
            }else{
                break;
            }
        }
```

```rust
let mut parite: bool = false;
let mut  salut: i32 = 0;
let mut error: bool = false;
for value: char in l1 {
    match value {
        'R' => if parite {
            parite = false;
        } else {
            parite = true;
        },
        'D' => if vec.len() == 0{
                error = true;
                println!("error");
                break;
            } else {
            if parite {
                vec.pop_back();
            }else{
                vec.pop_front();
            }
        }
        _ =>salut = salut +1,
    }
}
if !error {
    if vec.len() != 0 {
        if parite{
            for i: usize in 0..vec.len()/2{
                vec.swap(i, j: vec.len()-(i +1));
            }
        }
        let result: String = vec.into_iter().map(|i: usize| i.to_string() + ",").collect::<String>();
        println!("{}", format!("[{}]", &result[0..result.len() - 1]));
    }else{
        println!("[]");
    }

}
```

recount

```rust
use std::{io::{self, prelude::*}, vec, collections::HashMap};

Debug | ▶ Run | Debug
fn main() {
    let mut input: String = String::new();
    io::stdin().read_to_string(buf: &mut input).expect(msg: "Lecture Stdin");
    let lignes: Lines = input.lines();
    let mut vec: HashMap<&str, i32> = HashMap::new();

    for line: &str in lignes {
        if vec.contains_key(line){
            *vec.get_mut(line).unwrap() += 1;
        }else{
            vec.insert(k: line, v: 1);
        }
    }
```

Heure

```rust
fn main() {
    let mut input: String = String::new();
    io::stdin().read_to_string(buf: &mut input).expect(msg: "Lecture Stdin");
    let mut lignes: Lines = input.lines();

    let line1: &str = lignes.next().expect(msg: "ddsqdsq");
    let ligne1: usize = line1.parse::<usize>().expect(msg: "Salut");

    for i: usize in 0..ligne1{
        let line: Vec<i32> = lignes.next().unwrap().split(":").map(|x: &str| x.parse::<i32>().expect(msg: "Entier")).collect::<Vec<i32>>();
```

Pangramme

```rust
use std::{io::{self, prelude::*}, collections::{hash_map, HashMap, HashSet}, hash::Hash, primitive};
Debug | ▶ Run | Debug
fn main() {
    let mut input: String = String::new();
    io::stdin().read_to_string(buf: &mut input).expect(msg: "Lecture Stdin");
    let mut lignes: Lines = input.lines();
    let vecUpper: [char; 26] = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'
    let vec: [char; 26] = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'];
    for line: &str in lignes{
        let mut pangramme: bool = true;
        let mut result: String = String::from("");
        let mut set: HashSet<char> = HashSet::new();
        for car: char in line.chars(){
            set.insert(car);
        }
        for i: usize in 0..26{
            if !set.contains(&vec[i]) && !set.contains(&vecUpper[i]){
                pangramme = false;
                result = format!("{}{}", result, vec[i]);
            }
        }
        if pangramme{
            println!("PANGRAMME");
        }else{
        println!("{}", result);
        }
    }
```

Integerlist

```rust
use std::io::{self, prelude::*};

Debug
fn main() {
    // recuperation de l'entree standard
    let mut input = String::new();
    io::stdin().read_to_string(&mut input).expect("Lecture de stdin");
    let lines = input.lines();
    // vecteur contenant toutes les lignes de l'entree
    let mut data = vec![];
    // parcours des lignes et ajout dans le vecteur
    for line in lines.skip(1) {
        data.push(line);
    }

    for data in data.chunks(3) {
        if data[1] == "0" {
            if data[0].find("D").is_some() {
                println!("error");
            } else {
                println!("[]");
            }
            continue;
        }

        let values = &data[2][1..data[2].len() - 1].split(",").map(|x| x.parse::<i32>().expect("Entier")).collect::<Vec<i32>>();

        let mut bound : (usize,usize) = (0,0);

        let mut is_reverse : bool = false;

        for action in data[0].chars() {
            if action == 'R' { is_reverse = !is_reverse; }
            else {
                if is_reverse { bound.1 += 1; }
                else { bound.0 += 1; }
            }
        }

        if bound.0 > values.len() - bound.1 || bound.0 > values.len() || bound.1 > values.len() {
            println!("error");
            continue;
        }

        let result = &values[bound.0 .. values.len() - bound.1];

        if is_reverse {
            println!("{}", format!("{:?}",result.iter().rev().collect::<Vec<&i32>>()).split_whitespace().collect::<Vec<&str>>().join(""));
        } else {
            println!("{}", format!("{:?}",result).split_whitespace().collect::<Vec<&str>>().join(""))
        }
    }
}
```

5. Listes

5.1. Tableaux dynamiques (Vec<T>)

```
my_vector.sort()
my_vector.sort_by_key(|x| x.abs())
my_vector.reverse()
$ ["Bonjour", "Limoges", "!"].join("+")
> "Bonjour+Limoges+!"
```

5.2. Files (VecDeque<T>)

```
my_queue.push_back(element)
my_queue.back()
my_queue.back_mut()
my_queue.front()
my_queue.front_mut()
my_queue.pop_front()
```

5.3. Piles (Vec<T>)

```
my_stack.push(element)
my_stack.last()
my_stack.last_mut()
my_stack.pop()
```

5.4. Files de priorité (BinaryHeap<T

```
my_heap.push(element)
my_heap.peek()
my_heap.pop()
```

File de priorité ou tableau trié ? Utili
suppressions sont intercalées. Sinon, u

6. Problèmes sur les structure

Résolvez les problèmes Kattis suivants :

everywhere	I've Been Everywhere, Man
freefood	Free Food
recount	Recount
delimitersoup	Delimiter Soup
integerlists	Integer Lists
knigsoftheforest	Knigs of the Forest
conversationlog	Conversation Log
bank	Bank Queue
trendingtopic	Trending Topic
stockprices	Stock Prices

```
use std::collections::{HashSet, HashMap, VecDeque, BinaryHeap};
```

1. Chaînes de caractères (String et &str)

```
my_string.as_str()
my_str.to_string()
$ "  Bonjour  ".trim()
> "Bonjour"
```

2. Tuples

```
$ ("Bonjour", 3, vec![5.4, -3.2]).1     // type : (&str, i32, Vec<f64>)
> 3
```

3. Dictionnaires (tables de hachage)

3.1. Ensembles (HashSet<T>)

```
my_set.insert(element);
my_set.contains(element);
my_set.remove(element);
```

3.2. Tables d'association (HashMap<K, V>)

```
my_map.insert(key, value)
my_map.get(key)
my_map.get_mut(key)
my_map.remove(key)
my_map.keys()
my_map.values()
for (key, value) in my_map { ... }
```

4. Notion d'ordre

4.1. Ordre lexicographique

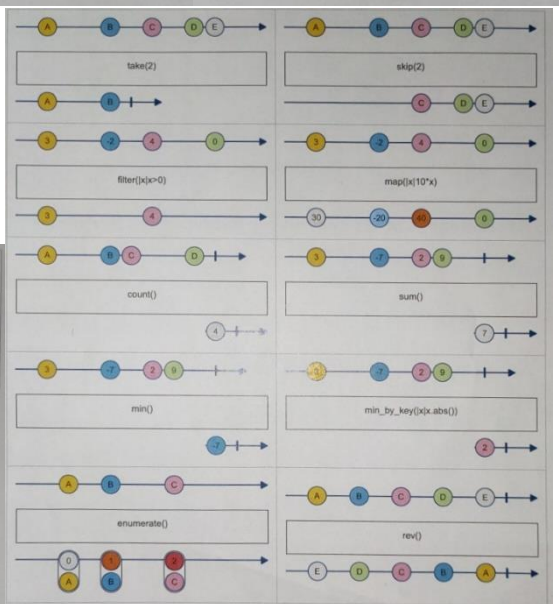
C'est l'ordre naturel «de gauche à droite» sur les tuples et les chaînes.

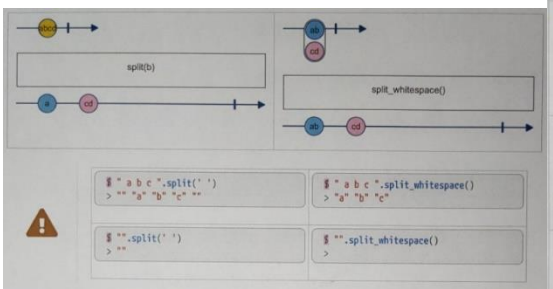
```
$ ("Bonjour", 8) < ("Bonsoir", 3)
> true
```

4.2. Renversement de l'ordre

```
use std::cmp::Reverse;
$ Reverse(3) < Reverse(4)       // type : Reverse<i32>
> false
```

Vec<T>		
Len()	récupérer le nombre d'éléments	O(1)
sort()		
Reverse()		





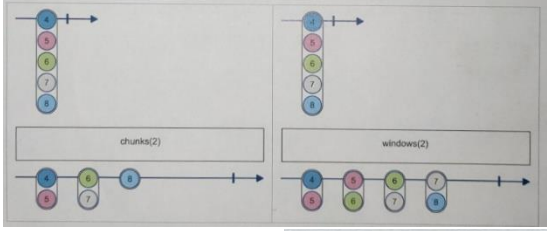

```
$ " a b c ".split(' ')
> "" "a" "b" "c" ""
```

```
$ " a b c ".split_whitespace()
> "a" "b" "c"
```

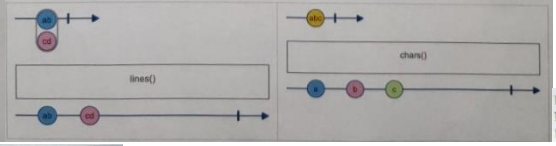
```
$ "".split(' ')
> ""
```

```
$ "".split_whitespace()
>
```

3. Opérateurs sur les vecteurs produisant des itérateurs



2. Opérateurs sur les chaînes produisant des itérateurs




```
let x = 3;              // i32
let ptr_x = &x;         // &i32
let y = *ptr_x;         // i32
```

Files

- extraction dans l'ordre d'insertion
- *FIFO: First In First Out*

VecDeque<T>

push_back(x)	insérer x en queue	O(1)
pop_front()	supprimer et récupérer l'élément x	O(1)

Piles

- extr
- *LIF*

Vec<T>

push(x)		
pop()		

Ensembles

- fonction : discriminer un ensemble d'objets
- pas de doublons

HashSet<K>

Insert(x)	Insérer x	O(1)
Contains(x)	x est-il présent ?	O(1)
Remove(x)	supprimer x	O(1)

```
use std::collections::HashSet;
let mut my_set = HashSet::new();
my_set.insert(18);
```

Tables d'association

- fonction : associer à chaque clé une valeur
- pas de doublons sur les clés

HashMap<K, V>

insert(k, v)	insérer (k, v)	O(1)
get(k)	Récupérer v associée à k	O(1)
remove(k)	supprimer (k, v)	O(1)

HashSet<T>, Vec<T>, VecDeque<T>

```
for element in my_collection.iter() {}
```

HashMap<K, V>

```
for (key, value) in my_map.iter() {}
```

```
retain(|&x| x % 2 == 0)
```

```
retain(|x| *x % 2 == 0)
```

Ne garder que les élém