GISTRE

Generics, Collections

-

Reduce code duplication with this simple trick !

Home exercises

- Anyone ?

```
> # Updating YOUR fork
> git remote add upstream https://github.com/cohenarthur/rust-gistre-workshop
> git fetch upstream # Fetch OUR repository's master branch
> git checkout master
> git rebase upstream/master # Rebase YOUR work on OUR updates
> # Tada!
```

Generics

Thanks to https://doc.rust-lang.org/book/ch10-01-syntax.html

```rust
fn largest_i32(list: &[i32]) -> &i32 {
    let mut largest = &list[0];

    for item in list {
        if item > largest {
            largest = item;
        }
    }

    largest
}
```

```rust
fn largest_char(list: &[char]) -> &char {
    let mut largest = &list[0];

    for item in list {
        if item > largest {
            largest = item;
        }
    }

    largest
}
```

```rust
fn main() {
    let number_list = vec![1, 15, 100, 85, 65];

    let result = largest_i32(&number_list);
    println!("The largest number is {}", result);

    let char_list = vec!['y', 'm', 'c', 'a'];

    let result = largest_char(&char_list);
    println!("The largest char is {}", result);
}
```

```
> cargo run
The largest number is 100
The largest char is y
```

```rust
fn largest<T>(list: &[T]) -> &T {
    let mut largest = &list[0];

    for item in list {
        if item > largest {
            largest = item;
        }
    }

    largest
}
```

```rust
fn main() {
    let number_list = vec![1, 15, 100, 85, 65];

    let result = largest(&number_list);
    println!("The largest number is {}", result);

    let char_list = vec!['y', 'm', 'c', 'a'];

    let result = largest(&char_list);
    println!("The largest char is {}", result);
}
```

```
error[E0369]: binary operation `>` cannot be applied to type `&T`
 --> src/main.rs:5:17
  |
5 |         if item > largest {
  |            ---- ^ ------- &T
  |            |
  |            &T
  |
help: consider restricting type parameter `T`
  |
1 | fn largest<T: std::cmp::PartialOrd>(list: &[T]) -> &T {
  |             ^^^^^^^^^^^^^^^^^^^^^^
  |
```

```rust
fn largest<T: PartialOrd>(list: &[T]) -> &T {
    let mut largest = &list[0];

    for item in list {
        if item > largest {
            largest = item;
        }
    }

    largest
}
```

Cheat sheet

```rust
struct AStruct<T> {
    x: T,
}
impl<T> AStruct<T> {
    fn New() -> AStruct<T> {
        AStruct { .. }
    }
}
fn parse_struct<T> (given: AStruct<T>){ .. }
```

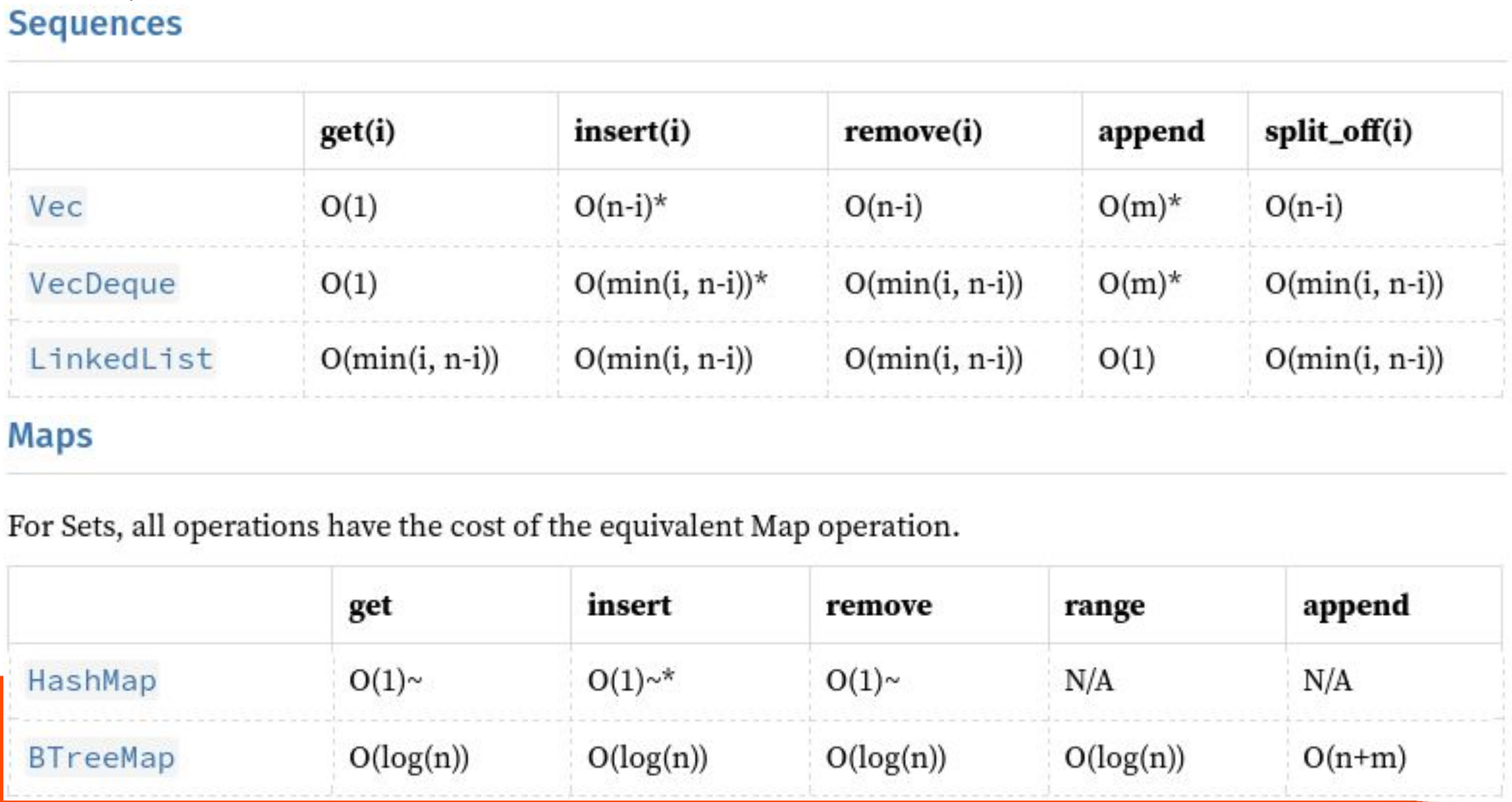
Multiple generic type parameters

```
struct AStruct<T, U> {
    x: T,
    y: U,
}
```

Collections

- Like the C++ STL
- 4 categories:
    - Sequences: Vec, VecDeque, LinkedList
    - Maps: HashMap, BTreeMap
    - Sets: HashSet, BTreeSet
    - Miscelleanous: BinaryHeap

Source: https://doc.rust-lang.org/std/collections/index.html



Sequences

	get(i)	insert(i)	remove(i)	append	split_off(i)
`Vec`	O(1)	O(n-i)*	O(n-i)	O(m)*	O(n-i)
`VecDeque`	O(1)	O(min(i, n-i))*	O(min(i, n-i))	O(m)*	O(min(i, n-i))
`LinkedList`	O(min(i, n-i))	O(min(i, n-i))	O(min(i, n-i))	O(1)	O(min(i, n-i))

Maps

For Sets, all operations have the cost of the equivalent Map operation.

	get	insert	remove	range	append
`HashMap`	O(1)~	O(1)~*	O(1)~	N/A	N/A
`BTreeMap`	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n+m)

Source: https://doc.rust-lang.org/std/collections/index.html

Collections are...

- Generic
- Efficient
- Easy to use

Collections are Idiomatic Rust:

- To create an empty one, you can use new()
- You can iterate over them (iter(), iter_mut()...)
- You can access their elements (collection[i] (kinda bad), collection.get(i), collection.get_mut(i), collection.insert(i, T))

```rust
let mut map = HashMap::new();
map.insert(10, String::from("Ten"));
map.insert(9, String::from("Nine"));

for (value, value_str) in map.iter() {
    println!("{}: {}", value, value_str);
}

println!("What's {} + {} ?", map[&9], map[&10]);
match map.get(&19) {
    Some(number_str) => println!("{}", number_str),
    None => println!("Idk 😭"),
}
```

Questions?