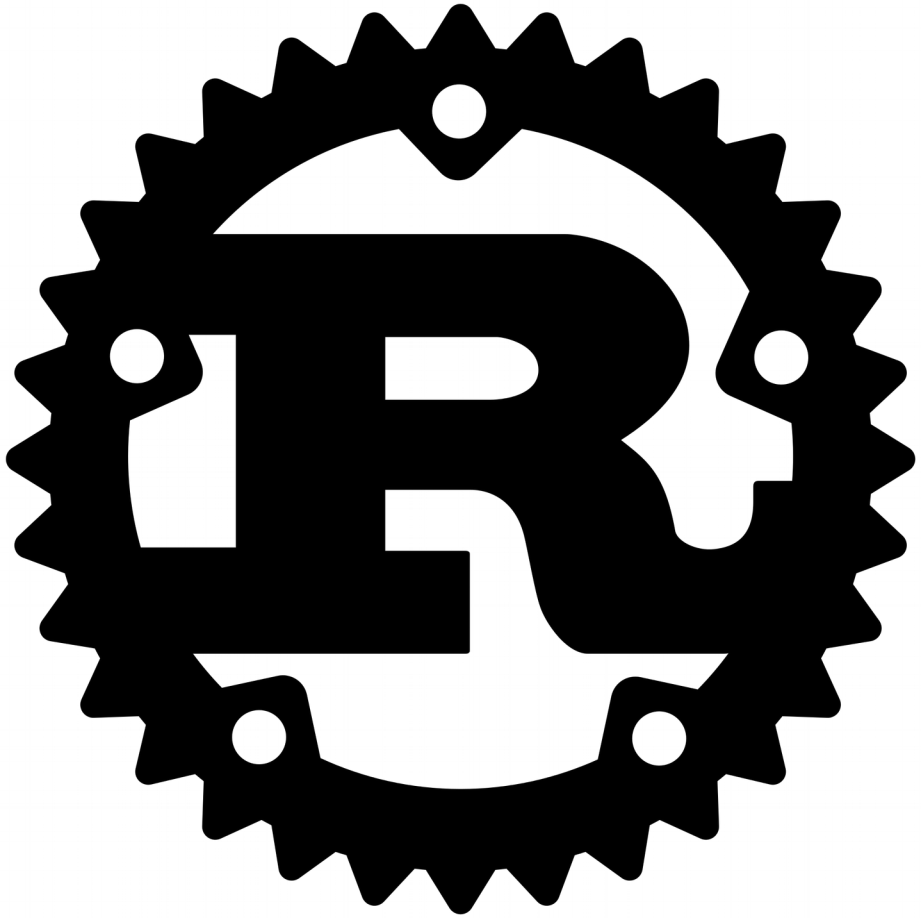

BONGRAND Guillaume, DIABIRA Binta, FLÉCHEUX Joan, HAUTIER Thomas

Plan

- Rust, c’est quoi ?
- Caractéristiques et avantages de Rust
- Dans quels cas utiliser Rust ?

Rust, c’est quoi ?

- Dérivé d’un projet personnel de Graydon Hoare, employé de Mozilla, débuté en 2006
- Soutenu par Mozilla depuis 2009, première alpha en 2012, première version stable en 2015
- rustc, un compilateur basé sur LLVM

Inspirations de Rust

Langage impératif avec éléments fonctionnels

- Typage fort, statique

```
//Java
final int M = 4;
int n = 5;

//Rust
let m: i32 = 4;
let mut n: i32  = 5;
```

- Inférence de type

- Pattern matching

```
let n = 7;

match n {
    1 => println!("Un"),
    2 | 4 => println!("Pair <5"),
    5...7 => println!(">4"),
    _ =>println!("3 ou >7 ,ou <1"),
}
```

Inspirations de Rust

Conçu pour être concurrent

```rust
use std ::thread;
use std::time::Duration;

fn main () {
    for i in 1..5 {
        thread::spawn(move || {
            println!("thread: {}", i);
            thread::sleep(Duration::from_millis(3));
        });
    }

    for i in 1..5 {
        println!("main: {}", i);
        thread::sleep(Duration::from_millis(3));
    }
}
```

```
ekrips@ekrips-laptop ~> rustc Test.rs
ekrips@ekrips-laptop ~> ./Test
main: 1
thread: 4
thread: 1
main: 2
thread: 2
main: 3
thread: 3
main: 4
```

rustc

Le compilateur est exigeant, mais un code qui compile a plus de chances de fonctionner correctement

```rust
let m: i32 = 54;
let n: i8 = 6;
println!("{}", m==n);
```

```rust
let m: i32 = 54;
let n: i32 = 6;
println!("{}", m==n);
```

```
ekrips@ekrips-laptop ~> rustc Test.rs
error[E0308]: mismatched types
 --> Test.rs:5:25
  |
5 |     println!("{:?}", m==n);
  |                         ^ expected i32, found i8

error: aborting due to previous error
```

```
ekrips@ekrips-laptop ~> rustc Test.rs
ekrips@ekrips-laptop ~> ./Test
false
```

Caractéristiques et avantages de Rust

- Memory safe
- Data race safe
- Macros
- Ecosystème solide

Rust est memory safe

- Principale raison : ownership
- Allocation de la mémoire le plus possible sur la pile
- Allocation sur le tas seulement quand on ne connaît pas la taille des données (par exemple vec)

Ownership

- Chaque variable a un unique propriétaire et hérite de sa portée

```
ekrips@ekrips-laptop ~/d/p/p/b/code> rustc Ownership.rs
error[E0382]: use of moved value: `v`
  --> Ownership.rs:11:37
   |
10 |     let v2 = v;
   |         -- value moved here
11 |     println!("v[0]: {}, v2[0]: {}", v[0], v2[0]);
   |                                     ^ value used here after move
   |
   = note: move occurs because `v` has type `std::vec::Vec<i32>`, which
 does not implement the `Copy` trait

error: aborting due to previous error
```

- Le compilateur vérifie si chaque variable est correctement utilisée puis détruite pendant la compilation

```
ekrips@ekrips-laptop ~/d/p/p/b/code> rustc Ownership.rs
error[E0425]: cannot find value `i` in this scope
 --> Ownership.rs:5:20
  |
5 |     println!("{}", i);
  |                    ^ not found in this scope

error: aborting due to previous error
```

Ownership

- Valeurs passées par référence mutable ou immutable ou par valeur

- Possible d’avoir plusieurs références immutables mais une seule mutable

Pile et tas

Par défaut, code sur la pile.

```
struct Foo {
    pub n: u32,
}

impl Foo {
    fn new () -> Foo {
        Foo {n: 24}
    }
}


fn main() {
    let foo = Foo::new();
    println!("{}", foo.n)
}
```

- Main alloue 4o sur la pile pour stocker foo (adresse : foo0)
- Main appelle Foo::new et lui donne foo0 en argument
- On entre dans Foo::new
- Foo::new alloue 4o sur la pile pour stocker un Foo (adresse : foo1)
- Foo::new stocke 24 dans foo1
- Foo::new copie foo1 dans foo0 puis termine
- On retourne dans main
- Main lit foo0 dans la pile et l’affiche

Pile et tas

- Chaque entité dont la taille n’est pas connue à la compilation est stockée dans le tas

```
struct Foo {
    pub v: Vec<u32>,
}

impl Foo {
    fn new () -> Foo {
        Foo {v:Vec::new()}
    }
}


fn main() {
    let mut foo = Foo::new();
    foo.v.push(24);
    println!("{}", foo.v[0])
}
```

Le fonctionnement identique, mais au lieu de copier la valeur dans foo1 on copie un pointeur vers l’emplacement de 24 sur le tas.

Data races

- Comme en Java, pas de data races inhérentes au langage

- Le seul partage de variable se fait par mutex ou variable atomique

Les macros

Morceaux de code proches des fonctions, mais "personnalisés"

```
macro_rules! sumIntArr {
    (
        $(
            $x:expr; [ $( $y:expr ),* ]
        );*
    ) => {
        &[ $($( $x + $y ),*),* ]
    }
}

fn main() {
    let a: &[i32]
        = sumIntArr!(10; [1, 2, 3];
            20; [4, 5, 6]);
    println!("a[0]: {}, a[1]: {}, a[2]: {}, a[3]: {}, a[4]: {}, a[5]: {}",
                a[0], a[1], a[2], a[3], a[4], a[5]);
}
```

```
ekrips@ekrips-laptop ~/d/p/p/b/code> rustc Macro_simple.rs
ekrips@ekrips-laptop ~/d/p/p/b/code> ./Macro_simple
a[0]: 11, a[1]: 12, a[2]: 13, a[3]: 24, a[4]: 25, a[5]: 26
```

Les macros

- Permet de ne pas se répéter (code générique)
- Utiliser un nombre variable d'arguments

```
macro_rules! calculate {
    // The pattern for a single `eval`
    (eval $e:expr) => {{
        {
            let val: usize = $e; // Force types to be integers
            println!("{} = {}", stringify!{$e}, val);
        }
    }};

    // Decompose multiple `eval`s recursively
    (eval $e:expr, $(eval $es:expr),+) => {{
        calculate! { eval $e }
        calculate! { $(eval $es),+ }
    }};
}

fn main() {
    calculate! {
        eval 1 + 2,
        eval 3 + 4,
        eval (2 * 3) + 1
    }
}
```

```
ekrips@ekrips-laptop ~/d/p/p/b/code> rustc Macro_calcul.rs
ekrips@ekrips-laptop ~/d/p/p/b/code> ./Macro_calcul
1 + 2 = 3
3 + 4 = 7
(2 * 3) + 1 = 7
```

Un écosystème solide

- Cargo
- Crates.io
- Rustbook & Rustonomicon
- Documentation solide

Dans quels cas utiliser Rust ?

- Conçu pour être massivement parallèle : navigateurs, traitement de données

- Applications à hautes performances : drivers, systèmes d’exploitation (plus généralement, Rust est adapté à la programmation système)

Questions ?