Rust : Ownership

Qualité de Développement — R5.A.08

C. Raïevsky

2023-2024

IUT A Valence
Université Grenoble Alpes

Département Informatique

BUT Informatique 3ème année

Ownership

Une variable a des droits sur les données qu'elle représente

Elle peut perdre ses droits

- Lorsque les données sont déplacées :
  - Vers une autre variable
  - Vers une fonction

Perte de droits - variable

```
1 fn main() {
2     let mut p1 = Point { x: 31.0, y: 33.0 };
3     let p2 = p1;                                  // Move occurs
4     p1.x = 42.0;                                  // Compile time error
5 }
```

Perte de droits - variable

```rust
fn main() {
    let mut p1 = Point { x: 31.0, y: 33.0 };
    let p2 = p1;                                    // Move occurs
    p1.x = 42.0;                                    // Compile time error
}
```

```
error[E0382]: assign to part of moved value: `p1`
 --> src/main.rs:4:5
  |
2 |     let mut p1 = Point { x: 31.0, y: 33.0 };
  |         ------ move occurs because `p1` has type `Point`, which does not implement the `Copy` trait
3 |     let p2 = p1;
  |              -- value moved here
4 |     p1.x = 42.0;
  |     ^^^^^^^^^^^ value partially assigned here after move
```

- L'assignation de p1 à p2 déplace les données → p1 n'a plus de droits
- p1 a perdu la propriété (*ownership*) des données

Perte de droits - fonction

```
1  fn main() {
2      let mut v = Vec2D {x: 12.0, y: 34.0};
3      let n = Vec2D::norm(v);              // Move occurs
4      v.x = 42.0;                          // Compile time error
5  }
6
7  impl Vec2D {
8      pub fn norm(v2d: Vec2D) -> f64 {
9          (v2d.x.powi(2) + v2d.y.powi(2)).sqrt()
10     }
11 }
```

Perte de droits - fonction

```
1  fn main() {
2      let mut v = Vec2D {x: 12.0, y: 34.0};
3      let n = Vec2D::norm(v);              // Move occurs
4      v.x = 42.0;                          // Compile time error
5  }
6
7  impl Vec2D {
8      pub fn norm(v2d: Vec2D) -> f64 {
9          (v2d.x.powi(2) + v2d.y.powi(2)).sqrt()
10     }
11 }
```

- ligne 3 : passage de v en paramètre → déplace les données
- v a perdu la propriété (*ownership*) des données

Déplacement des données par défaut

Lors d'une assignation ou d'un appel de fonction :

- Déplacement des données
- Perte de la propriété

Un des outils pour garantir l'utilisation sûre de la mémoire

Idée légitime

```
1  fn main() {
2      let mut v = Vec2D {x: 12.0, y: 34.0};
3      let n = Vec2D::norm(v);              // Move occurs
4      v.x = 42.0;                          // Compile time error
5  }
6
7  impl Vec2D {
8      pub fn norm(v2d: Vec2D) -> f64 {
9          (v2d.x.powi(2) + v2d.y.powi(2)).sqrt()
10     }
11 }
```

Comment rendre ce code fonctionnel ?

Deux options

Copier les données

- Utiliser une copie des données
- Nécessite :
  - Implémenter le trait "Copy" ou
  - Cloner explicitement les données

"Prêter" les données

- Utiliser une référence pour ne pas déplacer les données
- "*Borrowing*"

Copier les données - Trait Copy

Pour autoriser Rust à copier automatiquement vos données :

- Implémenter le trait "Copy" (et donc "Clone")

```
1
2 impl Clone for Vec2D {
3     fn clone(&self) -> Self {
4         Self { x: self.x, y: self.y }
5     }
6 }
7
8 impl Copy for Vec2D {
```

Copier les données implicitement - Trait Copy

```
1 fn main() {
2     let mut v = Vec2D {x: 12.0, y: 34.0};
3     let n = Vec2D::norm(v);                // Copy occurs implicitly
4     v.x = 42.0;
5 }
```

La variable v conserve la propriété de ses données

Copier les données explicitement - Trait Clone

```
1 fn main() {
2     let mut v = Vec2D {x: 12.0, y: 34.0};
3     let n = Vec2D::norm(v.clone());           // Explicit clone
4     v.x = 42.0;
5 }
```

La variable v conserve la propriété de ses données

Implémenter Copy gratuitement - `derive`

```rust
#[derive(Clone,Copy)]
pub struct Vec2D {
    pub x: f64,
    pub y: f64,
}
```

Pour les cas simples :

- Annotation "`derive`"
- Implémente les traits pour nous

Les types de base (i32, f64, etc.) implémentent le trait `Copy` (et donc `Clone`)

Copier les données - Pas toujours une bonne idée

```
1 struct PointList {
2     points: Vec<Point>,
3 }
```

En quoi consisterait la copie d'une variable de type `PointList` ?

- Duplication en mémoire de la liste des points ?
- S'il y a 200 000 points ?

Borrowing

Une variable peut prêter ses données

- En lecture seule
- En "mutable"

Lecture seule

- Plusieurs prêts simultanés possibles
- Impossible si un prêt "mutable" en cours

Mutable

- Si aucun autre prêt en cours,
- mutable ou non

Utilisation de références : opérateur "&"

```
1
2  fn main() {
3      let mut v = Vec2D {x: 12.0, y: 34.0};
4      let n = Vec2D::norm(&v);                // Borrow occurs
5      v.x = 42.0;                             // This is fine
6  }
7
8  impl Vec2D {
9      pub fn norm(v2d: &Vec2D) -> f64 {
10         (v2d.x.powi(2) + v2d.y.powi(2)).sqrt()
11     }
12 }
```

Référence "mutable"

```
1  fn main() {
2      let mut v = Vec2D {x: 12.0, y: 34.0};
3      v.normalize();                          // Mutable borrow occurs
4      v.x = 42.0;                             // This is fine
5  }
6
7  impl Vec2D {
8      pub fn normalize(&mut self) {    // Shortcut for:  (self: &mut Vec2D)
9          let n = self.norm();
10         self.x = self.x/n;
11         self.y = self.y/n;
12     }
13 }
```