**Seconde-session 2021-2022: L2 Programmation avancée**

Axel Viala <axel.viala@darnuria.eu>

7 juin 2022

Nom et Prénom : Numéro étudiant :

**Objectifs :** La clarté des réponses sera appréciée, veillez à écrire soigneusement. Les questions portent sur le langage Rust. Réponse sur une copie a part encouragée. 1 pts par question. Justification encouragée.

# 1 Généralités

1. Typage en Rust est : (plusieurs réponses valides() :

   - ◯ Dynamique à l'exécution
   - ◯ inféré à la compilation
   - ◯ Avec classes comme en Java
   - ◯ Explicite dans les fonctions, traits, structures et enumerations
   - ◯ Permet de l'héritage comme en Java
   - ◯ Permet des types sommes (énums)
   - ◯ Explicit partout comme en C
   - ◯ Traçage par le compilateur des partages de références
   - ◯ Pas typé

2. Une référence peut elle avoir une valeur `null` comme en C ? Justifiez ?

3. Dans le code ci dessous, qu'afficherait le programme s'il appelait `mystere(Some(1 + 1))` une fois ? Quelle est la valeur de retour de mystère avec cet appel ?

```rust
fn mystere(a: Option<i32>) -> i32 {
    match a {
        Some(n) => { println!("Mon mystère: {}", n); n + 42 },
        None => 0,
    }
}
```

4. Dans ce code, `m` est passé comment ?

   - ◯ Par déplacement *move*
   - ◯ Par référence *borrow* (mut/immutable)
   - ◯ Par copie *copy*

```rust
struct MystereStruct { secret: i32 }
fn mystere(m: &mut MystereStruct) {
    m.secret += 1;
}
```

5. Toute valeur par exemple une `struct A` hors type de base est :

- ◯ Pris par référence
- ◯ Déplacé
- ◯ Copié

6. Ce code peut-il compiler si non pourquoi ?

```
struct MystereStruct { secret: i32 }
fn add_thing(m: &mut MystereStruct, o: &MystereStruct, thing: i32) {
    m.secret += thing + o.secret;
}

fn main() {
    let mut a = MystereStruct { secret : 0 };
    add_thing(&mut a, &a, 1);
}
```

7. En Rust comment exprimez l'absence de quelque chose, en C on utiliserait probablement `null`, donnez un exemple de code, par exemple une structure personne qui peux avoir une date de naissance ou non.

8. Anatomie d'un code Rust : associez les termes suivants au code suivant :

- Opérateur d'addition
- Nom de variable de match
- Mot clef de pattern matching/filtrage
- Argument de fonction
- Type
- Mot clef de déclaration de fonction
- Constructeurs de variant de l'enum `Option`
- Bloc du corps de la fonction
- Nom de fonction
- Appel de fonction associée à un type
- Argument de fonction passé en appel
- Expression du if

```
fn inconnue(o: Option<i32>) -> Option<i32> {
    match o {
        Some(i) if i > 18 => Some(i + 50),
        _ => None,
    }
}
```

9. Que signifie `T` dans la signature de la fonction `fn mystere<T>(a: T)`.

10. Ce code peut-il compiler ? Justifiez.

```
.
fn mystere(mut a: Box<[i32]>, b: &i32) {
    a[0] += *b;
}

fn main() {
    // Box<T> n'implémente pas Copy
    let a = Box::new([1, 2, 3]);
    mystere(a, &a[0]);
    println!("{:?}", a);
}
```

11. Expliquez ce qu'est un `trait` en Rust donnez un exemple.

12. A quoi sert le type `Option<T>` et le type `Result<T, E>`

13. Proposez votre implémentation de `Result<T, E>` qui fait comme `Tesult`, Et implémenter la fonction `Result::map` qui dois faire comme la documentation de `Result::map` de la lib standard.

14. `bool` est il passé par copie ou par move par défaut.

15. Ce code comporte une ou plusieurs erreurs, laquelle justifiez. Que fait `?`. Justifiez.

```rust
fn mystere(a: Option<u32>) -> Option<u32> {
    let a = a?;
    None(a + 1)
}
```

16. Pour un point réalisez une implémentation du trait `Debug` et `Mul` entre deux `Point`.

```rust
struct Triangle {
    x: f32,
    y: f32,
}
```

## 2 Robot Quest

17. Implémentez une enum `Mouvement` qui dois représenter `Haut, Bas, Droite, Gauche, Avance`.
18. Programmez une structure représentant un personnage pouvant avoir une position dans l'espace et une orientation.
19. Réaliser une fonction qui si on lui passe un `&[Mouvement]` permet de déplacer un personnage.
20. Écrivez une fonction qui permet de réaliser des déplacements dans un monde en 2 dimensions, (a vous de choisir la structure) et que si le robot atteint un bord, son mouvement n'est pas effectué il n'avance pas et ne fait pas planter le programme, et du `Monde` est libre, un mouvement par tour de boucle jusque à ce que on ai parcouru tout les mouvements.
21. Ajouter dans votre représentation du monde des coffres en position (x, y) : (1, 1), (5, 6). Si le robot tombe sur un coffre il dois `println!("Coffre trouvé")` La representation des coffres est libre.