Ralph-Habib El-Mor

A24 | 5G4 - Technologies émergentes des applications

Groupe 01

Projet de Fin d’année

Le langage de programmation Rust (Note de Cours)

Travail Présenté à

Sylvain Labranche

Techniques de l’informatique

Collège Montmorency

16 décembre 2024

Table des Matières

[RUST 3](#_Toc185271127)

[Un peu de Contexte. 3](#_Toc185271128)

[Pourquoi Utiliser RUST 3](#_Toc185271129)

[Objectif du cours 3](#_Toc185271130)

[Premier Pas 4](#_Toc185271131)

[Structure et le print 4](#_Toc185271132)

[Type et Déclarations de variable 5](#_Toc185271133)

[Variable mutable 6](#_Toc185271134)

[println! 6](#_Toc185271135)

[Console input et output 6](#_Toc185271136)

[IF et Else If 8](#_Toc185271137)

[Les Boucles 9](#_Toc185271138)

[Les Tableaux 11](#_Toc185271139)

[Function et appel de function 12](#_Toc185271140)

[Lecture et Écriture dans un fichier 13](#_Toc185271141)

[Gestion des erreurs 15](#_Toc185271142)

[Gestion de mémoire et ownership 17](#_Toc185271143)

[Précision pour String ::from. 18](#_Toc185271144)

[Les Vecteur Précision 19](#_Toc185271145)

[Pourquoi utiliser Rust Dans le futur 20](#_Toc185271146)

[Sources/Bibliographie 21](#_Toc185271147)

# RUST

## Un peu de Contexte.

### Rust est un langage de programmation développer par Graydon Hoare un employé Firefox pendant son temps libre en 2006. Rust est influencer par divers langage (C++, Erlang, Haskell, Scala, OCaml, Scheme, Swift, C#, Alef, Limbo) et ce veux simple d’utilisation. Rust met l’accent sur la performance d’une application la sécurité du typage ainsi que sur la programmation de concurrence (l’utilisation de plusieurs cœurs).

## Pourquoi Utiliser RUST

### Premièrement, Rust est très rapide, performant et sécuritaire. Comme les langage C, il offre un contrôle sur les fonctions de base d’un système mais est plus simple d’utilisation. De plus, Il n’est pas obligatoire d’avoir Rust installer sur son ordinateur pour lancer une application . Il faut toute fois que l’os utiliser pour compiler le programme sois le même que pour l’exécuter.

## Objectif du cours

### Ce cours s’adresse à des étudiants en informatique ayant déjà des bases en programmation. Il pour but d’apprendre la spécificité de la syntaxe de Rust sans toute fois rentrer trop en profondeur dans la complexité. En suivant ce cours l’étudiant sera en mesure d’effectuer le laboratoire sans rencontrer de difficulté.

## Premier Pas

#### Dans ce cours, nous assumons que vous avez déjà les connaissances en de base informatique.

### Pour commencer Créer un Fichier sur votre bureau et ouvrer le avec vs-code. Par la suite utiliser la commande cargo init pour initialiser votre Fichier. Voici le Résultat. Vous pouvez exécuter la commande Cargo new “nom du projet”.

A screenshot of a computer

Description automatically generated

## Structure et le print

### Dans le Dossier ExempleCours Nous avons un dossier src ou ce trouve nos classes, un dossier Target ainsi que le fichier Cargo.toml qui contient des informations sur la version utiliser ainsi que les dépendances.

### Dans le répertoire src nous avons notre classe main.rs qui contient une Fonction main. Une Fonction dans Rust est Représenter par fn() ici fn main(){}. La fonction println! Est utilise pour afficher le message et le ; est mis à la fin.

### Pour Exécuter le programme vous pouvez utiliser le terminal vscode ou le command prompt. Naviguer vers la racine du fichier que vous voulez exécuter. Écrire ‘cargo build’. Quand c’est fini Naviguer vers le fichier target puis dans le fichier debug puis faite .\ ‘nomdevotrefichier’.exe pour exécuter votre fichier.

A black screen with white text

Description automatically generated

### NOTE

#### \* Si vous avez installé les extensions dans le guide vous pourrez exécuter sans passer par ces étapes en cliquant sur run/debug

A black screen with white text

Description automatically generated

## Type et Déclarations de variable

### Comme les autres langages de programmation, on peut assigner différents types aux variables que ce soit un int ou un string.

### Voici les différentes manières de déclarer une variable :

### Les entiers : let x : u32 =40(32bits); et let x : i8=40(8bits) ou i32, i64;

### Les floats : let x : f32 = 1.12; et let x : f64 = 7.78;

### Les boolean : let x :bool = true;

### Les chaines de charactère : let x : &str "Bonjour"

### Les tableaux : let x : [&str; 3] = [ "vert","jaune","rouge"];

### Toutefois, on n’est pas obligé de spécifier le type d’une variable pour la déclarer. On peut la déclarer simplement comme ça : let par exemple **let x = ¨Bonjour¨** le compilateur s’occupe d’assigner le type à la variable. On ne peut pas changer de type en réédifiant la variable x. En d’autres termes si x est déclarer comme un string, on ne pourra pas le changer pour un int plus tard dans le code.

A computer screen with green text

Description automatically generated

## Variable mutable

### Dans Rust il n’est pas possible d’affecter une différente valeur à une variable après qu’elle soit déclarer dans le code. Toutefois, si lors de la déclaration nous spécifions que la variable est mutable(let mut x = 2;) nous pouvons changer la valeur de cette variable dans le code. Le type de variable doit rester le même on ne peut pas passer d’un int à un String.

## println!

### Pour afficher la variable x que l’on a créée on utilise println!("{x} à tous"); On peut aussi l’afficher de cette manière : println!("{} à tous",x);

A computer screen shot of text

Description automatically generated

## Console input et output

### Dans Rust, comme dans d’autre langage de programmation on utilise des packages que l’on appelle crates. En ajoutant des packages on pourra se servir de fonction qui ne sont pas inclus dans la librairie de base. Pour nous permettre d’affecter manuellement des variable grâce a la console on utilisera la librairie io(input, output). On la déclare comme suit (use std::io;).

A screen shot of a computer code

Description automatically generated

### Dans ce programme, on demande à l’utilisateur de rentrer un nombre. On déclare une variable mutable que l’on défini comme un String car dans Rust, toute entrer dans la console est considéré comme un string. On appelle la méthode read\_line de stdin et on lui demande d’affecter la lecture a la variable intput. (Le expect est utiliser pour assigner un message d’erreur s’il en a une). Finalement on affiche le contenu de la variable.

### Le Programme n’est toute fois pas terminer car à quoi bon demander un nombre si on ne peut pas s’en servir pour faire un calcul. Pour ce faire, il va falloir convertir le string en int.

A screen shot of a computer code

Description automatically generated

### Ici j’ai déclaré une variable x comme étant un entier ensuite, j’ai utilisé la commande trim() sur input et puis parse pour convertir le string en int puis la méthode unwrap pour extraire la valeur. Finalement, j’ai additionné x à 100 et voici le résultat du code :

A black screen with white text

Description automatically generated

### N’oublier pas d’utiliser input.clear. Cela peut être utile dans une boucle qui sert a rentrer plusieurs input différente.

## IF et Else If

### Comme pour les autres langages de programmation Rust nous permet de faire des conditions if et else. Bien que la syntaxe soit différente, ils suivent la même logique.

A screen shot of a computer code

Description automatically generated

### Vous remarquerez que nous n’avons pas besoin d’utiliser un return ou encore de parenthèse dans pour les conditions

### Nous pouvons aussi stocker une valeur dans une variable en utilisant le if et else :

A computer screen shot of a program code

Description automatically generatedNoter toutefois qu’il faut comparer deux variables du même type on ne peut pas comparer des String avec des int ou des int avec des floats.

## Les Boucles

### Dans Rust, une manière de faire une boucle est d’utilisé Loop. Il suffit de déclarer dans le programme loop{} et tout ce que la parenthèse contient se répétera à l’infini. On peut cependant l’arrêter avec break; Voici un exemple :

A computer screen shot of a program code

Description automatically generated

### Dans Rust Loop peut aussi nous retourner une valeur :

A screen shot of a computer program

Description automatically generated

### Ici on arrête le programme grâce à break est on rajoute x après pour que la valeur de x soit initialisée dans la variable résultat;

### Un des cas ou on peut utiliser une boucle loop par exemple dans un jeu ou l’utilisateur dois deviner un nombre. L’utilisateur aura alors un nombre infini de chance de trouver le nombre.

### La boucle for est aussi présente dans Rust. Pour l’utiliser il y a deux façons différentes. :

A computer screen shot of text

Description automatically generated

### Ici nous déclarons la variables i puis un intervalle allant de 0 à 10(0..10). La boucle sera alors répétée 10 fois. Nous pouvons remplacer l’intervalle 0..10 par une variable déclarer à l’avance comme ceci :

A screen shot of a computer

Description automatically generated

Nous pouvons utiliser les boucles for pour parcourir un tableau (Aller voir la section tableau)

### Finalement, nous avons la boucle while et do while que l’on peut utiliser pour exécuter le code en attendant qu’une condition soit remplit.

A screen shot of a computer program

Description automatically generated

### \*À noter qu’il faut toujours utiliser une variable mutable.

Voici aussi un Exemple pour rentrer plusieurs variable dans un tableau :

## Les Tableaux

### Pour déclarer un tableau dans Rust, on peut le déclarer comme ceci : let x= [1,2,3,4,5];. Mais nous pouvons aussi les déclarer comme ça : let x = let x :[i32;5] = [1,2,3,4,5]; ou i32 est le type ici un int cela peut être un string &str et on vient après ; spécifier la longueur du tableau que l’on à déclarer dans ce cas 5; Pour obtenir une valeur d’une case encore la Rust n’est pas différent il suffit de faire println!("{}",x[0]);

### Pour parcourir ce tableau nous pouvons utiliser une boucle for ainsi que la fonction iter() qui comme son nom l’indique est un itérateur qui vas nous aider à parcourir le tableau.

A computer screen shot of a code

Description automatically generated with medium confidence

### Nous pouvons aussi utiliser une approche plus traditionnelle pour parcourir les tableaux :

A black background with colorful text

Description automatically generated with medium confidence

### Pour les tableaux nous avons aussi une autre façon d’utiliser la boucle for pour énumérer les informations contenues dans le tableau mais aussi leur index :

A black background with text and symbols

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated

## Function et appel de function

### Pour faire une fonctionner dans Rust on utilise l’abréviation fn puis on ajoute le nom de la fonction après par exemple fn addition(){}. Par convention on n’utilise pas de majuscule dans nos fonctions mais plutôt des \_ underscore pour séparer les mots.

### Bien sûr comme dans chaque langage, les fonctions utilisent souvent des paramètres. Lorsque l’on définit un paramètre dans Rust il faut préciser le type du paramètre.

A screen shot of a computer code

Description automatically generated



### Maintenant, si nous voulons qu’une fonction retourne une valeur il faudra spécifier en utilisant -> i32 juste avant la parenthèse. Puis dans la function main on déclare une variable et on assigne la fonction pour récupérer son résultat.

A computer screen with colorful text

Description automatically generated

### Noter que vous pouvez utiliser le return ou sans return dans la fonction addition.

## Lecture et Écriture dans un fichier

### Nous allons d’abord commencer par la lecture d’un fichier texte. Vous devez vous assurer que vous avez le fichier texte cible dans le répertoire Target de votre répertoire Rust. Dans le fichier main.rs (ou celui que vous utilisez) nous allons les package fs ::File qui nous permettra de manipuler le fichier ainsi que prelude pour lire le fichier.

A black background with text

Description automatically generated

A black screen with white text

Description automatically generated

### Dans mon exemple j’ai créé une variable fichier pour y stocker le fichier que je suis allé chercher en utilisant le chemin absolu vous pouvez essayer toutefois sans utiliser un chemin absolu. La méthode expect est utilisé afin de créer un message spécialiser pour savoir si le programme plante et à quelle étape il a planté pour mieux le déboguer. J’ai ensuite créé une variable texte pour stocker l’information grâce à la méthode file.read\_to\_string(&mut texte) et afficher le résultat;

### Pour l’écriture dans un fichier, nous avons déjà importer les packages important. Nous avons deux options, écrire dans un fichier existant ou en créer un nouveau.

### Commençons par la création d’un nouveau fichier :

A computer screen shot of a program

Description automatically generated

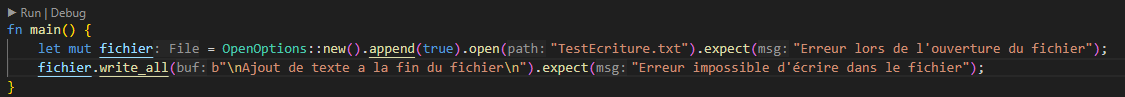
### Ici j’ai utilisé la méthode create pour créer un fichier txt à la base de mon projet, Grace à la méthode write\_all on peut écrire dans le fichier ce que l’on désir. Faire attention toutefois à l’extension du fichier pour qu’il soit bien la bonne puis avant le string à écrire si vous remarquer il y a la lettre b c’est pour spécifier que la chaine est en bytes. Il faut préciser car la méthode write\_all écrit les données en octets donc vous pouvez soit mettre b devant le string ou utiliser la méthode write\_all("Le fichier est creer".as\_bytes())

A screenshot of a computer

Description automatically generated



### Finalement nous somme pas obliger de créer un nouveau fichier pour écrire nos données à chaque nous pouvons simple les écrire dans un fichier déjà créer :



### Ici avec la méthode append(true) nous assurons que le fichier que l’on va ouvrir n’écrase pas le contenu mais l’ajoute à la fin puis nous procédons comme lors d’une ouverture et écriture de fichier classique.

A black screen with white text

Description automatically generated

## Gestion des erreurs

### Dans Rust nous avons principalement deux types d’erreur, des erreurs récupérables comme par exemple lors de l’ouverture d’un fichier qui n’existe pas. Nous avons aussi les erreurs irrécupérables qui nécessite l’arrêt total du programme par exemple lors d’une division par zéro.

### Pour gérer les erreurs récupérables nous pouvons utiliser Result qui vas nous permettre vérifier si l’opération a réussi ou si elle à échouer. La syntaxe de Result est composée de ok qui est utiliser pour les instructions lors du succès du programme ainsi que de Err pour les instructions en cas d’échec.

A computer screen shot of a program code

Description automatically generated

### Ici on utilise la méthode match pour capturer le résultat de la function lire\_fichier et dépendamment du Result on affichera le texte ou le message d’erreur dans la console.

### Nous pouvons aussi utiliser Option à la place d’ok nous avons some pour dire qu’il y a quelque chose dans de trouver s’il n’y a rien de trouver on utilise none pour retourner un message.

### Pour les erreurs irrécupérables nous pouvons utiliser les méthode panic! Qui va arrêter le programme immédiatement et afficher un message dans la console nous avons aussi les méthodes unwrap qui peut être utiliser pour récupérer une valeur contenue dans Result ou option mais il faut être sûr que Result et option contiennent quelque chose sinon unwrap panic est arrête le programme.

### Expect quant à lui envoi un message d’erreur personnaliser et panic si par exemple une variable ne contient rien.

### Panic! Arrête le programme complètement avec un message d’erreur on peut l’utiliser dans pour vérifier si une variable est null ou ne répond pas à nos critères par exemple et arrêter le programme après.

A screen shot of a computer code

Description automatically generated

## Gestion de mémoire et ownership

### Les langages de programmation ont besoin de gérer la mémoire qu’ils utilisent pour pouvoir fonctionner et le font généralement de deux manières différentes. Soit en effaçant la mémoire si elle n’est pas utilisée (java,python) mais le programme est plus lent par conséquent (Garbage collector). Soit en allouant la mémoire et en la suppriment explicitement (C++,assembly) plus rapide mais cela peut entrainer des fuites de mémoire si on oublie de libérer la mémoire. Le ownership dans Rust nous permet d’avoir une gestion de mémoire rapide tout en minimisant les erreurs.

### Le ownership est une fonctionnalité de Rust mais il aussi un concept en tant que tel. Une façon de programmer et d’organiser le code pour garantir qu’il soit efficace et sans erreur.

### Pour mieux comprendre l’ownership dans Rust nous devons d’abord connaitre la notion de pile et de tas car ils affectent directement le comportement du langage.

### La pile(Stack) : on utilise la pile pour stocker des valeurs dont on cannait la taille et dont la taille est fixe(Ne peut pas changer durant l’exécution du programme) les variables de la pile sont stockées en ordre la dernière stocker sera la première à être supprimer. Nous utilisons les méthode push et pop pour ajouter et retirer de la pile.

### Le tas(Heap) : lorsque l’on ne peut pas utiliser la pile pour stocker on peut utiliser le tas. Dans le tas, un espace de la taille de la valeur à stocker est réservé puis un pointeur est retourné avec la le chemin de l’espace.(On peut stocker le pointeur sur la pile).

### Exemple :

### Les variables x et y sont des données ayant une valeur fix ils seront donc stocker sur une pile. Cependant, pour s et vec, se sont des donner variable. Le contenu de s et vec est stocker sur le tas car ce dernier peut varier mais le pointeur qui point vers le tas est stocker dans la pile ainsi que la longueur du vecteur et du string.

### Les données stockées sur une pile sont plus rapides à accéder que se stockées sur un tas.

### Le ownership sert dans tout ça à gérer les données du tas. Il mémorise quand et par quoi les données sur le tas sont utilisées, réduis le nombre de copy de la même donner et détruit les données qui ne sont pas utiliser.

## Précision pour String ::from.

A black background with many colorful text

Description automatically generated with medium confidence

### Ici nous avons une variable let mut s = String ::from("Bonjour"). Déclarer le String au lieu de string va nous permettre de modifier ce dernier avec la méthode push\_str. Cela nous permet aussi d’utiliser la méthode clone sur le string pour le cloner et le mettre dans une nouvelle variable par exemple.

A computer screen shot of a math equation

Description automatically generated



A screen shot of a computer

Description automatically generated

### Bref grâce au String :: nous avons accès à une grande quantité de commande pour modifier le string.

## Les Vecteur Précision

### Pour pouvoir manipuler des tableaux dans Rust nous utilisons les vecteurs car ils sont de taille dynamique et nous permettrons par exemple de remplir un tableau grâce aux inputs en utilisant la méthode push. Exemple : A screen shot of a computer program Description automatically generated

### Ici nous déclarons un objet Vec majuscule comme le string nous pouvons le modifier comme String en utilisant le push par exemple. Dans ce code par exemple on utilise la taille dynamique du vecteur pour créer un tableau et l’afficher dans la console.

## Pourquoi utiliser Rust Dans le futur

### La question selon moi ne ce pose même pas. Rust est open-source, rapide et performent, sécuritaire , facile d’utilisation et il combine les forces de la plupart des langages de programmation traditionnel sans leur faiblesse. Je ne pense pas qu’il va remplacer tous les langages de programmation mais de plus en plus de compagnie utilise Rust pour faire des site web, coder des jeux ou le backend d’application de data science. Des compagnies comme Discord,Facebook,Microsoft et Amazon l’utilise aussi. Le programme n’est toutefois pas encore complet. Ces forces comme le ownership et le borrowing sont aussi des faiblesses car ces des concepts durs à comprendre. La librairie et les outils de Rust sont restreint et limiter et n’ont pas beaucoup de support offert. Finalement contrairement à java, python et c++ rust est un langage quand même récent et sa communauté bien qu’active reste petite comparer aux géants des langages de programmation.

## Sources/Bibliographie

### Articles Web Écrit par Cedric Sellmann

### 5 façons Que le langage de programmation Rust est utiliser

### <https://www.understandingrecruitment.com/knowledge-hub/blog/5-ways-rust-programming-language-is-used/>

### Pages Reddit sur les prédictions du futur de Rust

### <https://www.reddit.com/r/rust/comments/sdqwsz/what_can_we_predict_about_the_future_of_rust/>

### Article geeksforgeeks écrit par ayann07

### Le futur de Rust

### <https://www.geeksforgeeks.org/future-of-rust/>

### Blog Rust avec l’opinion de divers utilisateur sur le futur de Rust <https://users.rust-lang.org/t/how-do-you-see-the-future-of-rust/108108>

### Article Programming Geeks Club

### Pourquoi Rust n’est pas plus utiliser <https://programmingeeksclub.com/why-is-the-rust-programming-language-not-widely-adopted/>

### Video Youtube de Not A Bot

### Error Handling dans Rust

### <https://www.youtube.com/watch?v=zWNKQq1HSdg&ab_channel=NotABot>

### Video Youtube de Francesco Ciulla

### Ownership dans Rust

### <https://www.youtube.com/watch?v=9VBLOwmNE1g&t=791s&ab_channel=FrancescoCiulla>

### Vidéo YouTube de dcode

### Écriture dans un fichier

### <https://www.youtube.com/watch?v=OMgRkAhnpf8&ab_channel=dcode>

### Vidéo YouTube de dcode

### Lecture d’un fichier

### <https://www.youtube.com/watch?v=nQqraiMymcU&ab_channel=dcode>

### Vidéo YouTube de Tech With Tim

### Functions,Expression et statements tutoriel Rust

### <https://www.youtube.com/watch?v=APrANyLHCtQ&ab_channel=TechWithTim>

### Vidéo YouTube de dcode

### La boucle For Rust

### <https://www.youtube.com/watch?v=gtoj6vOeb1A&ab_channel=dcode>

### Vidéo YouTube de dcode

### La boucle infinit Rust

### <https://www.youtube.com/watch?v=dUW7FB_ajMs&ab_channel=dcode>

### Vidéo YouTube de dcode

### La boucle While

### <https://www.youtube.com/watch?v=OaEXxpELfvw&ab_channel=dcode>

### Vidéo YouTube de dcode

### Les tableaux

### <https://www.youtube.com/watch?v=cH6Qv47MPwk&ab_channel=dcode>

### Video Youtube de Francesco Ciulla

### Control flow If,Match loop while et for

### <https://www.youtube.com/watch?v=2w_FTiD74JI&t=518s&ab_channel=FrancescoCiulla>

### Vidéo YouTube de Tech With Tim

Les condition et le control flow

### <https://www.youtube.com/watch?v=MOa7ulhNYc0&t=1s&ab_channel=TechWithTim>

### Vidéo YouTube de Tech With Tim

### Les constantes et le shadowing

### <https://www.youtube.com/watch?v=xYgfW8cIbMA&ab_channel=TechWithTim>

### Vidéo YouTube de Tech With Tim

### Introduction a la programmation de rust

### <https://www.youtube.com/watch?v=T_KrYLW4jw8&t=1s&ab_channel=TechWithTim>

### Documentation rust Sur les String et leur Structure

### <https://doc.rust-lang.org/std/string/struct.String.html>

### Documentation de Rust

### <https://doc.rust-lang.org/book/>