

u/the1ExuberantRaptor sur Reddit

Rust

en programmation orientée objets

Firmin LAUNAY

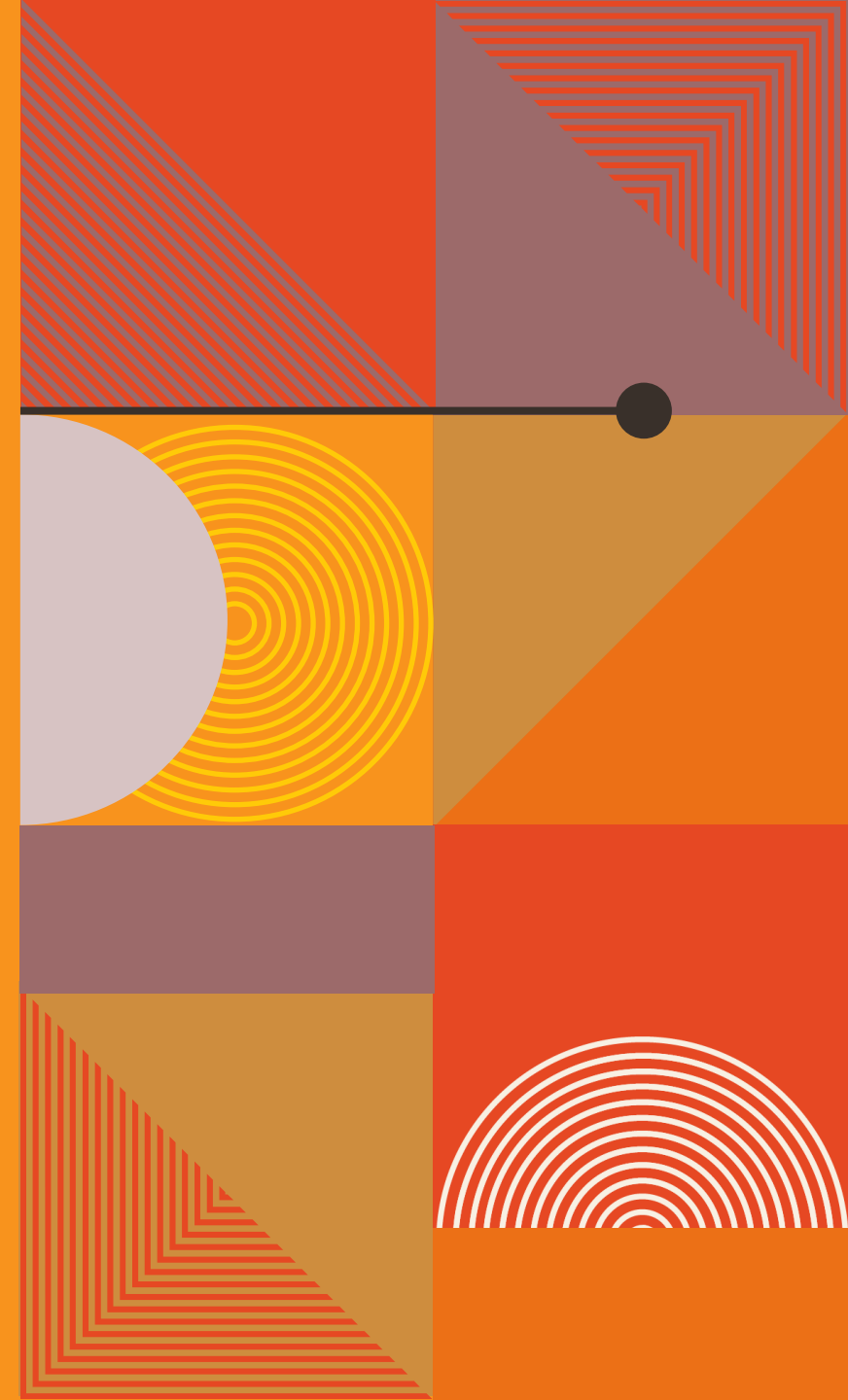
Firmin_Launay@etu.u-bourgogne.fr

20 mars 2024

Polytech Dijon – semestre 6

SOMMAIRE

- I. À propos de Rust
- II. Rust en pratique
- III. Retour d'expérience sur Rust





À PROPOS DE RUST



HISTOIRE DE RUST

- **2006** : Graydon HOARE, employé de Mozilla Resarch crée Rust en tant que projet personnel.
- **2009** : Mozilla parraine officiellement le projet.
- **2015** : La première version stable est publiée le 15 mai. Le langage est rapidement adopté par de grandes entreprises.
- **2022** : Rust est pris en charge dans le développement du noyau Linux.
- **2023** : Microsoft annonce l'intégration de Rust dans le noyau Windows.

CARACTÉRISTIQUES DE RUST

- Langage **compilé**
- Langage en règle générale **performant**
- **Typage sûr**
- Support des **opérations concurrentes** (threads)
- Assure la **sécurité mémoire** par le **suivi des objets** tout au long de leur vie (c'est la raison même pour laquelle Rust a été créé)
- « langage **hyperexpressif** moderne, mais de **bas niveau**, deux propriétés habituellement antinomiques »

B. Vigneron, « Rust, mes premières impressions »,
20 mai 2021, via Medium

SYNTAXE DE RUST

VARIABLES

Initialisées avec le mot-clef **let**. Par défaut immuables, sauf si on y ajoute le mot-clef **mut**. Mode de fonctionnement assez rare.

STRUCTURES DE CONTRÔLE

Utilisables comme des expressions classiques.

Exemple de code valide :

```
let k = if (i < 0) { 0 } else { 1 };
```

Équivalent en C :

```
int k = (i < 0) ? 0 : 1
```

CONSTANTES

Définies avec le mot-clef **const**. Valeur fixée à la compilation et ne peuvent donc pas dépendre d'une autre variable non-**const**.

CLASSES ?

enum, **struct**, **impl**, **trait**.
Fonctionnement un peu différent par rapport à C++, Java ou encore Python. Pas un langage pensé pour la POO au départ.

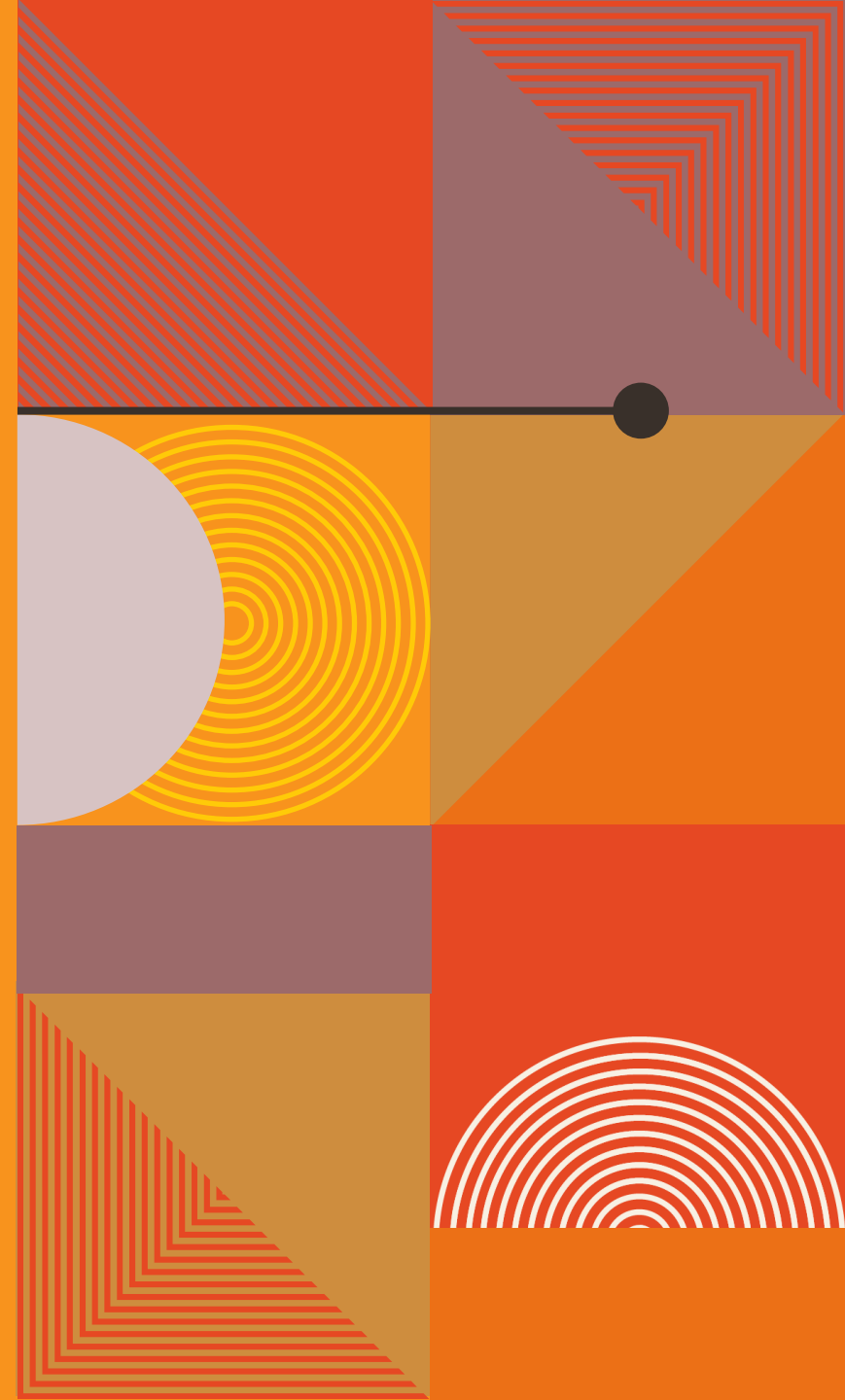
CAS D'UTILISATION DE RUST

- Développement de systèmes d'exploitation
- Développement de logiciels proches du système
- Développement de *backends* performants
- Développement d'outils en ligne de commande
- Développement web (surtout côté serveur)

bas niveau

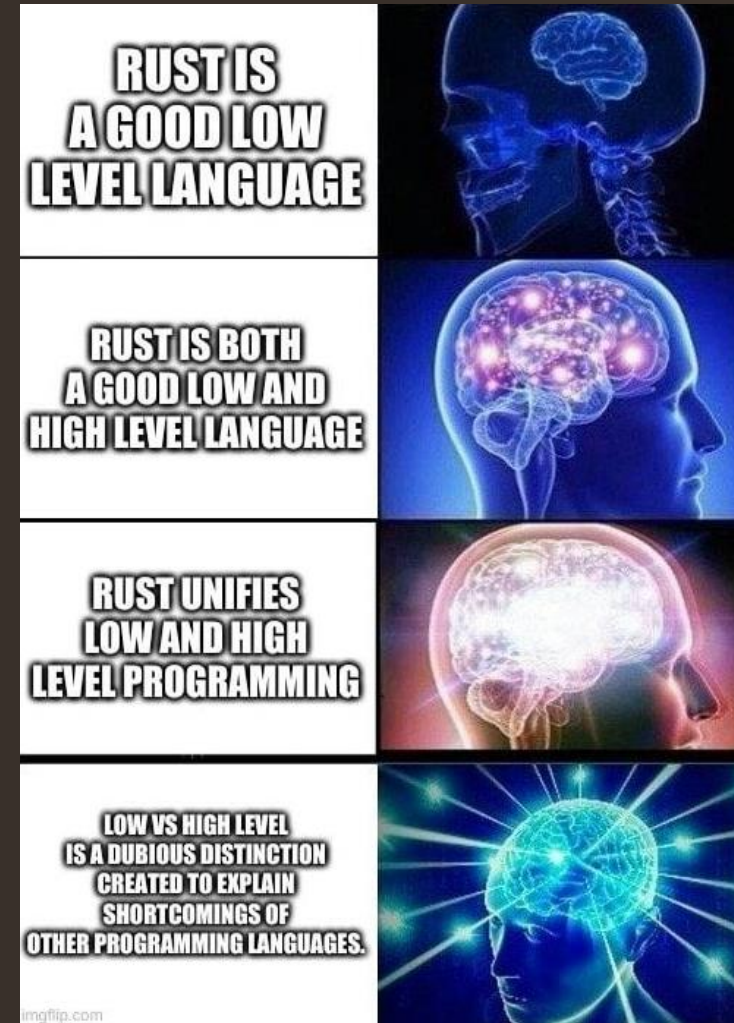


haut niveau



RUST, UN LANGAGE VERSATILE ?

En fait, Rust semble adapté pour de la programmation bas niveau comme haut niveau. Est-ce alors un « langage à tout faire » ?



u/dpc_pw sur Reddit

RUST, UN LANGAGE VERSATILE ?

Théoriquement, pesons le pour et le contre...

POINTS POSITIFS

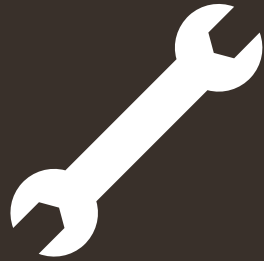
- *Memory safe* (même sans ramasse-miettes)
- Produit du code machine proche du matériel (au moins autant que C)
- Système de packaging très performant
- Génération de documentation facile

POINTS NÉGATIFS

- Survole uniquement les concepts de POO
- Structure assez différente par rapport aux langages qu'on connaît : apprentissage relativement difficile
- Système de packaging très complexe
- Compilation assez lente



RUST EN PRATIQUE

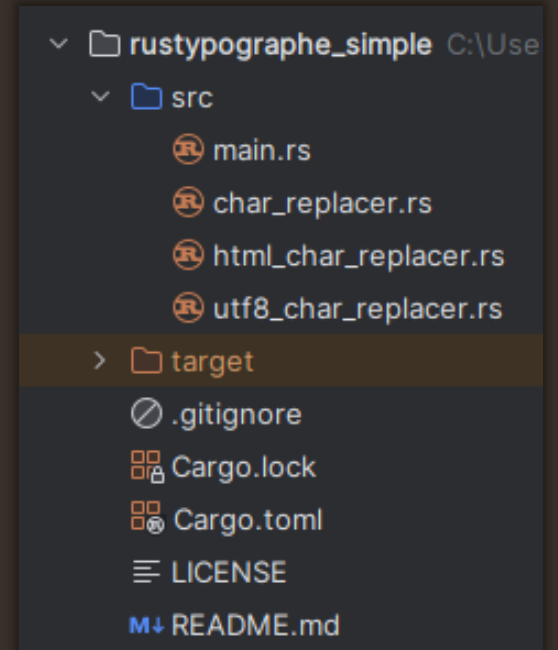


INSTALLATION DE RUST

- Sur Windows, utilisation du gestionnaire de paquets Chocolatey.

`choco install rust`

- Deux principaux éléments : le compilateur, `rustc` et le gestionnaire de projets et de dépendances, `cargo`.





APPLICATION EN RUST

- Correcteur typographique assez basique.
- Corrige des chaînes UTF-8 et du HTML.
- Utilisation complète des concepts de POO en Rust, même lorsque c'est un peu « excessif ».
- Développé sous forme d'« exécutable » (devrait être une bibliothèque).



<https://github.com/Firmin-ESIREM/rustypographe>

DÉMONSTRATION

Base-text:·La·nuit·était·calme,·éclairée·seulement·par·la·lueur·de·la·lune,·lorsque·soudain,·un·bruissement·dans·les·buissons·attira·mon·attention...·Intrigué,·je·m'approchai·lentement,·le·cœur·battant,·me·demandant·ce·qui·pouvait·bien·se·cacher·dans·les·"ténèbres"·nocturnes.¶

Corrected, UTF-8:·La·nuit·était·calme,·éclairée·seulement·par·la·lueur·de·la·lune,·lorsque·soudain,·un·bruissement·dans·les·buissons·attira·mon·attention...·Intrigué,·je·m'approchai·lentement,·le·cœur·battant,·me·demandant·ce·qui·pouvait·bien·se·cacher·dans·les·«ténèbres»·nocturnes.¶

Corrected, HTML:·La·nuit·était·calme,·éclairée·seulement·par·la·lueur·de·la·lune,·lorsque·soudain,·un·bruissement·dans·les·buissons·attira·mon·attention…·Intrigué,·je·m'approchai·lentement,·le·cœur·battant,·me·demandant·ce·qui·pouvait·bien·se·cacher·dans·les·« ténèbres »·nocturnes.¶

Remarque : Fonctionnalités implémentées assez limitées dans le cas présent.

```

use duplicate::duplicate_item;

pub trait FindAndReplace {
    fn find_and_replace_occurrences(&self, text: String) -> String;
}

pub(crate) trait Replacer {
    fn to_replace(&self) -> &Vec<String>;
    fn replacement(&self) -> &str;
    // fn occurrences(&self) -> &Vec<Vec<u8>>;
}

#[derive(Debug, Clone, Copy)]
impl Replacer for SpecificReplacer {
    fn to_replace(&self) -> &Vec<String> {
        &self.to_replace
    }

    fn replacement(&self) -> &str {
        &self.replacement
    }
}

impl<T> FindAndReplace for T where T: Replacer {
    fn find_and_replace_occurrences(&self, text: String) -> String {
        let mut result = text;
        for element_to_replace in self.to_replace() {
            result = result.replace(element_to_replace, self.replacement());
        }
        return result;
    }
}

/* -----
| ELLIPSES REPLACER |
----- */

pub(crate) struct EllipsesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl EllipsesReplacer {
    pub(crate) fn new(replacement_str: &str) -> EllipsesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("..." .to_string());
        to_replace.push("..." .to_string());
        return EllipsesReplacer {
            to_replace,
            replacement,
        };
    }
}

/* -----
| QUOTES REPLACER |
----- */

pub(crate) struct OpeningQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl OpeningQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> OpeningQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("\"" .to_string());
        to_replace.push("'" .to_string());
        to_replace.push("`" .to_string());
        to_replace.push("@raquo;" .to_string());
        return OpeningQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

pub(crate) struct ClosingQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl ClosingQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> ClosingQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("\"" .to_string());
        to_replace.push("'" .to_string());
        to_replace.push("`" .to_string());
        to_replace.push("@raquo;" .to_string());
        return ClosingQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

```

- Une classe est définie par un **struct**.
- On greffe des méthodes à cette structure en utilisant **impl**.
- On a un semblant d'héritage en implémentant notre structure en partant d'un **trait**.

```

use duplicate::duplicate_item;

pub trait FindAndReplace {
    fn find_and_replace_occurrences(&self, text: String) -> String;
}

pub(crate) trait Replacer {
    fn to_replace(&self) -> &Vec<String>;
    fn replacement(&self) -> &str;
    // fn occurrences(&self) -> &Vec<Vec<u8>>;
}

#[duplicate_item(SpecificReplacer; [EllipsesReplacer]; [OpeningQuotesReplacer]; [ClosingQuotesReplacer])]
impl Replacer for SpecificReplacer {
    fn to_replace(&self) -> &Vec<String> {
        &self.to_replace
    }

    fn replacement(&self) -> &str {
        &self.replacement
    }
}

impl<T> FindAndReplace for T where T: Replacer {
    fn find_and_replace_occurrences(&self, text: String) -> String {
        let mut result = text;
        for element_to_replace in self.to_replace() {
            result = result.replace(element_to_replace, self.replacement());
        }
        return result;
    }
}

/* -----
| ELLIPSES REPLACER |
----- */

pub(crate) struct EllipsesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl EllipsesReplacer {
    pub(crate) fn new(replacement_str: &str) -> EllipsesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("..."_to_string());
        to_replace.push("..."_to_string());
        to_replace.push("..."_to_string());
        return EllipsesReplacer {
            to_replace,
            replacement,
        };
    }
}

/* -----
| QUOTES REPLACER |
----- */

pub(crate) struct OpeningQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl OpeningQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> OpeningQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("\""_to_string());
        to_replace.push("'"_to_string());
        to_replace.push("@"_to_string());
        to_replace.push("@"_to_string());
        return OpeningQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

pub(crate) struct ClosingQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl ClosingQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> ClosingQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("\""_"_to_string());
        to_replace.push("'"_"_to_string());
        to_replace.push("@_"_to_string());
        to_replace.push("@_"_to_string());
        return ClosingQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

```

```

use duplicate::duplicate_item;

pub trait FindAndReplace {
    fn find_and_replace_occurrences(&self, text: String) -> String;
}

pub(crate) trait Replacer {
    fn to_replace(&self) -> &Vec<String>;
    fn replacement(&self) -> &str;
    // fn occurrences(&self) -> &Vec<Vec<u8>>;
}

#[duplicate_item(SpecificReplacer; [EllipsesReplacer]; [OpeningQuotesReplacer]; [ClosingQuotesReplacer])]
impl Replacer for SpecificReplacer {
    fn to_replace(&self) -> &Vec<String> {
        &self.to_replace
    }

    fn replacement(&self) -> &str {
        &self.replacement
    }
}

impl<T> FindAndReplace for T where T: Replacer {
    fn find_and_replace_occurrences(&self, text: String) -> String {
        let mut result = text;
        for element_to_replace in self.to_replace() {
            result = result.replace(element_to_replace, self.replacement());
        }
        return result;
    }
}

```

```

use duplicate::duplicate_item;

pub trait FindAndReplace {
    fn find_and_replace_occurrences(&self, text: String) -> String;
}

pub(crate) trait Replacer {
    fn to_replace(&self) -> &Vec<String>;
    fn replacement(&self) -> &str;
    // fn occurrences(&self) -> &Vec<Vec<u8>>;
}

pub(crate) trait EllipsesReplacer: Replacer {
    fn new(replacement_str: &str) -> EllipsesReplacer;
}

impl Replacer for SpecificReplacer {
    fn to_replace(&self) -> &Vec<String> {
        &self.to_replace
    }

    fn replacement(&self) -> &str {
        &self.replacement
    }
}

impl<T> FindAndReplace for T where T: Replacer {
    fn find_and_replace_occurrences(&self, text: String) -> String {
        let mut result = text;
        for element_to_replace in self.to_replace() {
            result = result.replace(element_to_replace, self.replacement());
        }
        return result;
    }
}

/* -----
| ELLIPSES REPLACER |
----- */

pub(crate) struct EllipsesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl EllipsesReplacer {
    pub(crate) fn new(replacement_str: &str) -> EllipsesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("..."&#x20;to_string());
        to_replace.push("..."&#x20;to_string());
        to_replace.push("..."&#x20;to_string());
        return EllipsesReplacer {
            to_replace,
            replacement,
        };
    }
}

/* -----
| QUOTES REPLACER |
----- */

pub(crate) struct OpeningQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl OpeningQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> OpeningQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        return OpeningQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

pub(crate) struct ClosingQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl ClosingQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> ClosingQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        return ClosingQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

```

```

/* -----
| ELLIPSES REPLACER |
----- */

```

```

pub(crate) struct EllipsesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl EllipsesReplacer {
    pub(crate) fn new(replacement_str: &str) -> EllipsesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("..."&#x20;to_string());
        to_replace.push("..."&#x20;to_string());
        return EllipsesReplacer {
            to_replace,
            replacement,
        };
    }
}

```

```

/* -----
| QUOTES REPLACER |
----- */

```

```

pub(crate) struct OpeningQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl OpeningQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> OpeningQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        to_replace.push("&#x20;"&#x20;to_string());
        return OpeningQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

```




```

use duplicate::duplicate_item;

pub trait FindAndReplace {
    fn find_and_replace_occurrences(&self, text: String) -> String;
}

pub(crate) trait Replacer {
    fn to_replace(&self) -> &Vec<String>;
    fn replacement(&self) -> &str;
    // fn occurrences(&self) -> &Vec<Vec<u8>>;
}

pub(crate) trait SpecificReplacer: Replacer {
    fn to_replace(&self) -> &Vec<String> {
        &self.to_replace
    }

    fn replacement(&self) -> &str {
        &self.replacement
    }
}

impl<T> FindAndReplace for T where T: Replacer {
    fn find_and_replace_occurrences(&self, text: String) -> String {
        let mut result = text;
        for element_to_replace in self.to_replace() {
            result = result.replace(element_to_replace, self.replacement());
        }
        return result;
    }
}

/* -----
| ELLIPSES REPLACER |
----- */

pub(crate) struct EllipsesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl EllipsesReplacer {
    pub(crate) fn new(replacement_str: &str) -> EllipsesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("..."&str.to_string());
        to_replace.push("..."&str.to_string());
        return EllipsesReplacer {
            to_replace,
            replacement,
        };
    }
}

/* -----
| QUOTES REPLACER |
----- */

pub(crate) struct OpeningQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl OpeningQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> OpeningQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("\"\"&str.to_string());
        to_replace.push("'&str.to_string());
        to_replace.push("&str.to_string());
        to_replace.push("&str.to_string());
        return OpeningQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

pub(crate) struct ClosingQuotesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

impl ClosingQuotesReplacer {
    pub(crate) fn new(replacement_str: &str) -> ClosingQuotesReplacer {
        let replacement = replacement_str.to_string();
        let mut to_replace = Vec::new();
        to_replace.push("\"\"&str.to_string());
        to_replace.push("'&str.to_string());
        to_replace.push("&str.to_string());
        to_replace.push("&str.to_string());
        return ClosingQuotesReplacer {
            to_replace,
            replacement,
        };
    }
}

```

```

use duplicate::duplicate_item;

pub trait FindAndReplace {
    fn find_and_replace_occurrences(&self, text: String) -> String;
}

pub(crate) trait Replacer {
    fn to_replace(&self) -> &Vec<String>;
    fn replacement(&self) -> &str;
    // fn occurrences(&self) -> &Vec<Vec<u8>>;
}

#[duplicate_item(SpecificReplacer; [EllipsesReplacer]; [OpeningQuotesReplacer]; [ClosingQuotesReplacer])]
impl Replacer for SpecificReplacer {
    fn to_replace(&self) -> &Vec<String> {
        &self.to_replace
    }

    fn replacement(&self) -> &str {
        &self.replacement
    }
}

impl<T> FindAndReplace for T where T: Replacer {
    fn find_and_replace_occurrences(&self, text: String) -> String {
        let mut result = text;
        for element_to_replace in self.to_replace() {
            result = result.replace(element_to_replace, self.replacement());
        }
        return result;
    }
}

/* -----
| ELLIPSES REPLACER |
----- */

pub(crate) struct EllipsesReplacer {
    to_replace: Vec<String>,
    replacement: String,
}

```



RETOUR D'EXPÉRIENCE SUR RUST

RETOUR SUR RUST

- En POO, structure beaucoup plus complexe qu'avec des langages plus « classiques ».
- Syntaxe un peu déroutante au début, mais somme toute assez logique et consistante.
- Côté *memory safe* appréciable (possibles soucis détectés à la compilation).



FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

 ▶ [ONCD](#) ▶ [BRIEFING ROOM](#) ▶ [PRESS RELEASE](#)

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report [here](#)

WASHINGTON – Today, the White House Office of the National Cyber Director (ONCD) released a report calling on the technical community to proactively reduce the attack surface in cyberspace. ONCD makes the case that technology manufacturers can prevent entire classes of vulnerabilities from entering the digital ecosystem by adopting memory safe programming languages. ONCD is also encouraging the research community to address the problem of software measurability to enable the development of better diagnostics that measure cybersecurity quality.

THE WHITE HOUSE

Administration Priorities The Record Briefing Room Español MENU

FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

According to experts, both memory safe and memory unsafe programming languages meet these requirements. At this time, the most widely used languages that meet all three properties are C and C++, which are not memory safe programming languages. Rust, one example of a memory safe programming language, has the three requisite properties above, but has not yet been proven in space systems. Further progress on development toolchains, workforce education, and fielded case studies are needed to demonstrate the viability of memory safe languages in these use cases. In the interim, there are other ways to achieve memory safe outcomes at scale by using secure building blocks. Therefore, to reduce memory safety vulnerabilities in space or other embedded systems that face similar constraints, a complementary approach to implement memory safety through hardware can be explored.



Merci

de votre attention

Firmin LAUNAY
Firmin_Launay@etu.u-bourgogne.fr