.UBAfiuba
FACULTAD DE INGENIERÍA

Guía de Ejercicios 2: Ownership en Rust

75.42 / 95.08 Taller de Programación I - 1er C 2021
Ing. Pablo Deymonnaz

Ejercicio 1

Analizar las siguientes porciones de código y responder si el mismo compila o no. Explicar por qué sí o por qué no.

Si no se compila, ¿qué podrías cambiar para que compile?

```
fn main() {
        let mut s = String::from("hola");
        let ref1 = &s;
        let ref2 = &ref1;
        let ref3 = &ref2;
        s = String::from("chau");
        println!("{}", ref3.to_uppercase());
}
```

Listing 1: Fragmento de código 1

```
fn drip_drop() -> &String {
    let s = String::from("hello world!");
    return &s;
}
```

Listing 2: Fragmento de código 2

```
fn main() {
        let s1 = String::from("hola");
        let mut v = Vec::new();
        v.push(s1);
        let s2: String = v[0];
        println!("{}", s2);
}
```

Listing 3: Fragmento de código 3

Ejercicio 2 - diff

Encontrar la diferencia entre dos archivos es un problema que es bastante conocido y estudiado.

La mayoría de las implementaciones usan el algoritmo de Myers, en este ejercicio, haremos que calcule la subsecuencia común más larga entre los dos archivos con el algoritmo LCS y use esa información para calcular su diferencia.

Este ejercicio se divide en hitos a cumplir.

Leer los dos archivos en dos vectores de líneas

En este hito, se debe implementar la función *read_file_lines* la cual debe tomar como parámetro la ruta al archivo y devolver un vector conteniendo las líneas del archivo.

Implementar el algoritmo LCS - Longest Common Subsequence

Longest Common Subsequence es un algoritmo conocido: dadas dos secuencias, ¿cuál es la subsecuencia más larga que aparece en ambas?

Si las secuencias de caracteres son **a b c d** y **a d b c**, la subsecuencia común más larga es **a b c**, porque estos caracteres aparecen en ambas secuencias en ese orden (notar que la subsecuencia no necesita ser consecutiva, sino que debe estar en orden).

Cuando se hace el diff entre dos archivos, queremos determinar cuáles línas deben ser agregadas o eliminadas entre ellos. Para lograr esto, necesitamos identificar las línas que son comunes entre ambos archivos. Esto se enmarca en lo que se conoce como un problema LCS[1]: tenemos las dos secuencias de líneas y queremos encontrar la mayor subsecuencia de línas que aparecen en ambos archivos; estas líneas son la que no fueron modificadas y las otras líneas son las que fueron agregadas o eliminadas.

La solución incluye completar una grilla con los largos de subsecuencias. El siguiente es un fragmento de pseudocódigo que se puede usar como base para reimplementar en Rust:

```
let X and Y be sequences
let m be the length of X, and let n be the length of Y

C = grid(m+1, n+1)
// recordar que .., es inclusivo para el límite inferior, pero excluye al superior
for i := 0..m+1
    C[i,0] = 0
for j := 0..n+1
    C[0,j] = 0
for i := 0..m
    for j := 0..n
        if X[i] = Y[j]
            C[i+1,j+1] := C[i,j] + 1
        else
            C[i+1,j+1] := max(C[i+1,j], C[i,j+1])

return C
```

Listing 4: Pseudocódigo del algoritmo LCS

Usar el LCS para construir el diff

Implementar e invocar al siguiente pseudocódigo para imprimir el diff:

[1] Hay un buen video explicativo de LC en: *https://www.youtube.com/watch?v=NnD96abizww*

```
// C es la grilla computada por lcs()
// X e Y son las secuencias
// i y j especifican la ubicacion dentro de C que se quiere buscar cuando
//    se lee el diff. Al llamar a estar funcion inicialmente, pasarle
//    i=len(X) y j=len(Y)
function print_diff(C, X, Y, i, j)
        if i > 0 and j > 0 and X[i-1] = Y[j-1]
                print_diff(C, X, Y, i-1, j-1)
                print "  " + X[i-1]
        else if j > 0 and (i = 0 or C[i,j-1] >= C[i-1,j])
                print_diff(C, X, Y, i, j-1)
                print "> " + Y[j-1]
        else if i > 0 and (j = 0 or C[i,j-1] < C[i-1,j])
                print_diff(C, X, Y, i-1, j)
                print "< " + X[i-1]
        else
                print ""
```

Listing 5: Pseudocódigo para construir el diff

Requerimientos no funcionales

Los siguientes son los requerimientos no funcionales para la resolución de los ejercicios:

- El proyecto deberá ser desarrollado en lenguaje Rust, usando las herramientas de la biblioteca estándar.
- No se permite utilizar *crates* externos. El único crate autorizado a ser utilizado es *rand*[2] en caso de que se quiera generar valores aleatorios.
- El código fuente debe compilarse en la versión stable del compilador y no se permite utilizar bloques unsafe.
- El código deberá funcionar en ambiente Unix / Linux.
- El programa deberá ejecutarse en la línea de comandos.
- La compilación no debe arrojar *warnings* del compilador, ni del linter *clippy*.
- Las funciones y los tipos de datos (*struct*) deben estar documentadas siguiendo el estándar de *cargo doc*.
- El código debe formatearse utilizando *cargo fmt*.
- Las funciones no deben tener una extensión mayor a 30 líneas. Si se requiriera una extensión mayor, se deberá particionarla en varias funciones.
- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

[2] https://crates.io/crates/rand