Grep Rústico

Introducción

grep(1) (*Globally Search For Regular Expression and Print out*) es una herramienta de la línea de comando de los sistemas Linux y Unix para la búsqueda de líneas que coincidan con un patrón específico en un archivo o grupo de archivos.

La sintaxis básica de *grep* requiere dos parámetros fundamentales: una expresión regular y la ruta hacia el archivo (relativa o absoluta).

```
$ grep regular_expression path/to/file
```

Ejemplo

Supongamos que tenemos un archivo llamado *fruits.txt* cuyo contenido es el siguiente:

```
banana
apple
orange
pineapple
melon
watermelon
```

y queremos buscar aquellas frutas cuyo nombre contenga la palabra "apple". Si ejecutamos *grep* en la terminal, obtendremos el siguiente resultado por salida estándar:

```
$ grep "apple" fruits.txt
apple
pinapple
```

Si quisiéramos buscar aquellas cuyo nombre contenga "melon":

```
$ grep "melon" fruits.txt
melon
watermelon
```

Por ultimo, si quisieramos buscar aquellas cuyo nombre contenga "apple" o "melon", podemos utilizar el operador de alternancia de la siguiente manera:

```
$ grep "apple|melon" fruits.txt
apple
pinapple
melon
watermelon
```

Expresiones regulares

Una expresión regular (o *regex*) es una cadena de caracteres basadas en reglas sintácticas que permiten describir secuencias de caracteres. Las expresiones regulares se construyen análogamente a las expresiones aritméticas, mediante el uso de varios operadores para combinar expresiones más pequeñas.

Una expresión regular puede estar compuesta, o bien solo por caracteres normales (como "melon"), o bien por una combinación de caracteres normales y metacaracteres. Los **metacaracteres** describen ciertas construcciones o disposiciones de caracteres.

Retomando el ejemplo anterior

¿Cómo podríamos hacer para obtener aquellas frutas que **finalicen** con la letra *e*?

El metacaracter **$** nos permite matchear expresiones al final de una línea. Haciendo uso de ello, podríamos construir la siguiente expresión regular y aplicarla a nuestra lista de frutas:

```
$ grep "e$" fruits.txt
apple
orange
pineapple
```

Ejercicio

El ejercicio consta de dos partes:

Como **primera parte**, el objetivo será investigar sobre el funcionamiento del comando grep, y sobre la sintaxis y funcionamiento de las expresiones regulares. Como punto de partida, se recomienda utilizar el manual oficial de Linux.

```
$ man grep
```

Esto permitirá familiarizarse con la lectura de documentación técnica, además de fomentar la investigación de detalles que resulten poco claros al lector en la documentación oficial.

Como **segunda parte**, el objetivo será implementar una versión *rústica* de la variante de grep: **egrep**.

egrep deberá ser invocado solo con la expresión regular y la ruta del archivo a evaluar (semejante a los ejemplos brindados previamente). Estos serán pasados como argumentos de línea de comando. El resultado deberá ser impreso por terminal.

Se deberá implementar funcionalidad para la expresiones que contengan:

- Caracteres normales
- Metacaracteres:
  - Period: **.**
  - Bracket expresion: **[]**
  - Bracket expresion negada: **[^]**
  - Character Classes:
    - [:alnum:]
    - [:alpha:]
    - [:digit:]
    - [:lower:]
    - [:upper:]
    - [:space:]
    - [:punct:]
  - Anchoring: **^**, **$**
  - Repetition: **?**, *****, **+**, **{n}**, **{n,}**, **{,m}**, **{n,m}**

Además, su implementación deberá permitir la **concatenación**, la **alternancia**, y la **precedencia** de expresiones regulares. Por ejemplo las siguientes expresiones regulares deberán estar soportadas:

```
ab.cd
ab.*cd
a[bc]d
ab{2,4}cd
abc|de+f
la [aeiou] es una vocal
la [^aeiou] no es una vocal
hola [[:alpha:]]+
[[:digit:]] es un numero
el caracter [[:alnum:]] no es un simbolo
hola[[:space:]]mundo
[[:upper:]]ascal[[:upper:]]ase
es el fin$
```

Algunas consideraciones

- Dado que estamos implementando la versión extendida de grep (egrep), algunos caracteres normales pasan a ser metacaracteres, adquiriendo un significado especial.

Para interpretar estos caracteres como caracteres normales, deben ser antecedidos por un backslash ****.
    - Ejemplo: si quisiéramos matchear el caracter literal **?**, debemos poner **\?**.
- Se debe tener especial cuidado a la hora de probar el comando grep (o egrep) fuera de un ambiente Unix/Linux, dado que podría no funcionar como es esperado debido a diferencias en el encoding de caracteres (por ejemplo, ejecutandolo en *Windows Subsystem for Linux*).
- El archivo de entrada posee formato ASCII. Los Strings de Rust y los &str son UTF-8. Para esta implementación solo brindaremos soporte a caracteres ASCII.
- **En caso de tener dudas sobre los aspectos del enunciado, siempre se debe consultar a los docentes para aclararlas.**

Nota importante: La entrada y salida del programa deben ser las mismas que las del comando **egrep**. Esto permitirá la correcta verificación del funcionamiento del programa, no cumplir con este punto requerirá la re-entrega inmediata del ejercicio. El uso de colores en la salida es opcional y se considerara un "bonus point".

Restricciones

- Escribir el programa sin utilizar .unwrap() o .expect(). Todo caso deberá manejarse ideomaticamente con las estructuras y funciones brindadas por el lenguaje.
- No se permite que el programa lance un panic!().
- No se permite utilizar la función exit(). Se deberá salir del programa finalizando el scope de la función main.
- No se permite utilizar el módulo mem para la manipulación de memoria.
- Está estrictamente prohibido el llamado al comando **grep** o **egrep** dentro del programa, o el llamado a cualquier otro comando.
- Para realizar un uso adecuado de memoria y respetar las reglas de ownership se deberá evitar el uso de .clone() y .copy().
- Todo el programa puede ser resuelto con lo aprendido en clase hasta la presentación de este ejercicio. No se espera que se utilicen estructuras relacionadas a concurrencia o redes para resolución de este ejercicio.

Requerimientos no funcionales

Los siguientes son los requerimientos no funcionales para la resolución del proyecto:

- El proyecto deberá ser desarrollado en lenguaje Rust 1.76, usando las herramientas de la biblioteca estándar.
- El proyecto deberá realizarse de manera individual. Cualquier tipo de copia significa la expulsión automática de la materia.

- No está permitido el uso de código generado por ninguna IA, ni copiar código de soluciones existentes en internet.
- Se deben implementar tests unitarios y de integración de las funcionalidades que se consideren más importantes.
- No se permite utilizar crates externos.
- El código fuente debe compilarse en la versión estable 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 documentados 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.

Fechas de entrega

Primer entrega: Miércoles 3 de Abril de 2024 hasta las 18hs.

No cumplir con la primer entrega imposibilitará la continuidad en la materia

Luego de la primer entrega se harán las correcciones correspondientes y se podrá volver a entregar el ejercicio en dos oportunidades más. La forma de entrega se comunicará por el canal de avisos.