

Rust the Forth

Donde `stack overflow` no es la solución, sino el problema...

Introducción

Forth es un lenguaje de programación orientado a pilas diseñado por [Charles H. Moore](#) (no, no es el de la Ley de Moore, ese es [Gordon Moore](#)) y utilizado por primera vez en el año 1970. A lo largo de su historia, Forth está presente principalmente en desarrollos orientados a aplicaciones en astronomía (como el sistema de vuelo de la sonda [Philae](#)) y en sistemas embebidos, debido a su diseño apropiado para la limitada memoria de los microcomputadores de la época. Adicionalmente, a lo largo de la década de los 80, Electronic Arts ha publicado múltiples juegos desarrollados en Forth, incluyendo [Worms?](#), [Lords of Conquest](#) y [Starflight](#), el cual, según sus desarrolladores, fue elegido debido a que "era más fácil que assembly y mas compacto".

Estructura del lenguaje

Forth basa su funcionamiento en el uso de stacks de datos y en la [Notación Polaca Inversa](#). Este tipo de notación postfix le otorga facilidad para el parseo y extension del lenguaje.

Ejemplo 1

Si quisiésemos computar el valor de la siguiente expresión aritmética: $3 * (25 + 10)$, utilizando notación polaca inversa, escribiríamos:

```
25 10 + 3 * CR .
```

Al ejecutar esta expresión primero se lee el número 25 y se pusha en la pila, luego ocurre lo mismo con el número 10. Ejecutar un valor numérico sin contexto es igual a pushar su valor a la pila.

BASE
25
10

Luego aplicamos la suma (**+**), la cual popea los primeros dos elementos de la pila, aplica la operacion, y luego pushea el resultado a la pila.

BASE
35

Pusheamos el 3 a la pila.

BASE
35
3

Aplicamos la multiplicación (analoga en funcionamiento a la suma), y pusheamos el resultado a la pila.

Luego, aplicamos la funcion **CR** (carriage return), que aplica un salto de linea a la salida estandar.

BASE
105

Finalmente, aplicamos la funcion **.**, la cual popea el primer elemento del stack y lo imprime por salida estandar.

BASE

El resultado de ejecutar el anterior programa sería el siguiente (noten la linea vacía al comienzo):

```
<cr>
105
```

Definicion de nuevas palabras (words)

Forth permite la definición de nuevas palabras (en adelante, denominadas *words*) para la extensión del lenguaje a traves de la siguiente sintaxis:

```
: <word-name> <word-body> ;
# Es importante la definición de espacios entre words
```

Para su almacenamiento, Forth implementa un diccionario de palabras que mapea nombres a código Forth ejecutable. Normalmente la implementación de este diccionario se lleva a cabo mediante un árbol de listas enlazadas, aunque no será obligatorio utilizar este tipo de dato (TDA) para este ejercicio.

Forth permite adicionalmente la redefinición tanto de words definidas en runtime como de words y operadores (+, -, *, etc.); no así la redefinición de números.

```
: 4 5 ; # Esto no es válido
```

Aclaración: las words son case-insensitive.

Ejemplo 2

Si quisiésemos definir una word MAX, el cual dado dos números retorne el máximo entre ellos:

```
: MAX OVER OVER < IF SWAP THEN DROP ;
```

```
10 20 MAX
```

Pusheamos 10 y 20 a la pila. Aplicamos la word MAX.

BASE
10
20

Aplicamos OVER dos veces. OVER toma el segundo elemento mas cercano a la parte superior de la pila, y lo duplica, posicionandolo en la parte superior.

BASE
10
20
10
20

Aplicamos <. Si bien en Forth no existen los booleanos en sí, por convención el 0 es tratado como falso, y cualquier otro valor como verdadero (aunque canónicamente, se considera al -1 como verdadero).

Aplicamos IF, el cual popea el primer valor de la pila y:

- Si el valor es verdadero, ejecuta la siguiente secuencia de words.
- Si es falso, ignora el resto de las palabras hasta encontrarse con la palabra THEN.

Dado que en nuestro caso es verdadero (-1),

BASE
10
20
-1

Aplicamos SWAP, el cual invierte la posición de los primeros dos elementos de la pila, y,

BASE
20
10

Aplicamos DROP, el cual remueve el primer elemento de la pila, dejando finalmente el valor máximo en la punta de la pila.

BASE
20

Ejercicio

El ejercicio propuesto es implementar una versión en Rust de un intérprete del estándar Forth-79.

Operaciones soportadas

El intérprete deberá soportar las siguientes operaciones:

- Aritmética de enteros: `+`, `-`, `*`, `/`.
- Manipulación de stack: `DUP`, `DROP`, `SWAP`, `OVER`, `ROT`.
- Definición de words.
- Generación de output: `.`, `EMIT`, `CR`, `." "`.
- Operaciones booleanas: `=`, `<`, `>`, `AND`, `OR`, `NOT`.
- Evaluación de condicionales: `IF ... THEN`, `IF ... ELSE ... THEN`.

- Aclaracion: **IF** consume el ultimo elemento del stack.

La pila de datos del intérprete debera estar diseñada para almacenar enteros (con signo) de 16 bits.

Respecto de la sintaxis, seguiremos una serie de reglas sencillas (no necesariamente las que sigue el estandar de Forth, pero mas que suficientes para este ejercicio):

Un numero es una secuencia de uno o mas digitos ASCII; una word es una secuencia de una o mas letras, digitos, simbolos y puntuaciones que no conforme un numero.

Manejo de errores

Haremos un manejo sencillo de errores: en caso de error, imprimiremos el error por stdout (sin aplicar saltos de linea) y terminaremos la ejecucion del script .fth. Algunos errores comunes pueden ser:

- **stack-underflow**: cuando una operación intenta poppear un elemento de una pila vacía.
- **stack-overflow**: cuando una operación intenta pushear un elemento a una pila que se encuentra en su capacidad máxima de memoria.
- **invalid-word**: cuando se trata de definir una word invalida, por ejemplo: **: 1 1 ;**.
- **division-by-zero**: cuando se trata de dividir por cero.
- **?**: cuando el interprete no puede hallar la definición de la word evaluada.

En estos casos, el mensaje de error debe ser tal cual el especificado previamente.

En caso de otros errores que no pertenezcan a ninguna de estas categorías, se debera imprimir un error custom descriptivo.

Formato de input

Se llamará a nuestro programa pasándole por primer parámetro la ruta al modulo .fth a evaluar. Opcionalmente, se puede pasar el tamaño del stack (en bytes) a reservar por el interprete; el tamaño por defecto sera de 128 KB.

```
cargo run -- ruta/a/main.fth stack-size={{size}}
```

Formato de output

El output se realizara a través de salida estandar (STDOUT, no STDERR), y se espera que **solamente** se impriman aquellas cosas producidas por las words que generan output. Cada operación de output debe estar separada por whitespace, a excepcion de **CR** que, por definición, imprime un *carriage return*, aunque en nuestro caso imprimiremos un *line feed* (**\n**).

Además, al finalizar la ejecución, se debe escribir en un archivo del directorio actual llamado **stack.fth** el stack restante de la ejecución, en orden de inserción (los elementos más antiguos primero). Si el archivo ya existe, debe sobrescribirse.

Por ejemplo, el siguiente código:

```
1 2 3 4 5
. . CR .
```

Imprimirá por STDOUT lo siguiente:

```
5 4
3
```

Y escribirá en el archivo **stack**:

```
1 2
```

Mas ejemplos

Hello world

```
: HELLO CR ." Hello, World!" ;
HELLO
```

```
# Output
<cr>
Hello, World!
```

Valor absoluto de un numero

```
: NEGATE -1 * ;  
: ABS DUP 0 < IF NEGATE THEN ;  
-30 ASB .
```

```
# Output  
30
```

¿Número es par?

```
: EVEN? DUP 2 / 2 * = ;  
30 EVEN? .  
15 EVEN? CR .
```

```
# Output  
-1 <cr>  
0
```

Stack Underflow

```
1 1  
. . cr .
```

```
# Output  
1 1  
stack-underflow
```

Recursos útiles

- [Forth-79](#): publicación del Forth Standards Team.
- [Easy Forth](#): un ebook diseñado para enseñar Forth de una manera sencilla mediante ejemplos, escrito por Nick Morgan.

Restricciones

- Escribir el programa sin utilizar `.unwrap()` o `.expect()`. Todo caso de error deberá manejarse ideomáticamente 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.
- 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 (ni se acepta) 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 la última version estable de Rust (compilador versión 1.85), 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.

IMPORTANTE: no cumplir con todas las restricciones y requisitos no funcionales implica la reentrega automática del ejercicio. Se debe prestar minuciosa atención a cada uno de los detalles.

Fechas de entrega

Primer entrega: Lunes 31 de Marzo de 2025 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.