UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

TP-N1

Autómata Finito Determinístico

Sintaxis y semántica de los lenguajes

Docentes:

- Ing. Roxana Leituz

Alumnos:

- Mateo Fernandez Cruz
- Marcos Ezequiel Benegas
- Ignacio Comes Kinderknecht
- Ariel Perez
- Braian Amador

Fecha de entrega:

- 1 de Octubre de 2024

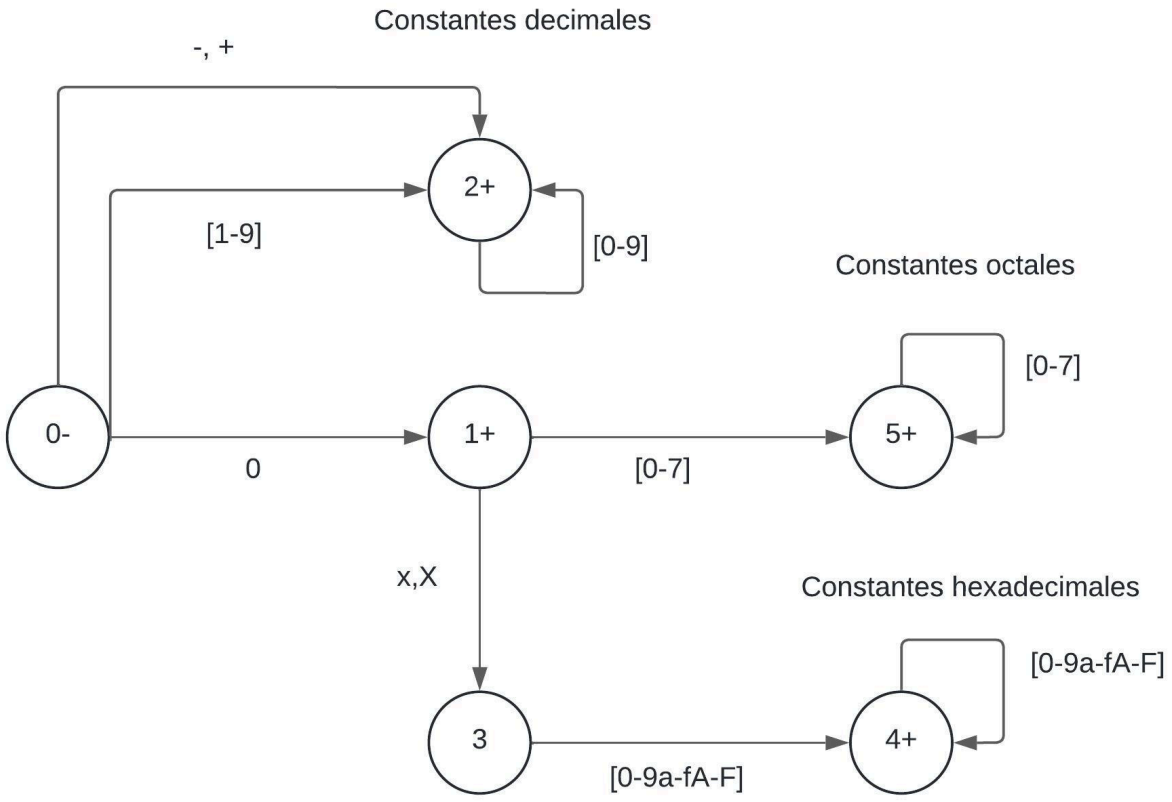
Descripción del proyecto

Implementación de un autómata para el reconocimiento y conteo de constantes numéricas en diversas bases (decimales, octales y hexadecimales) dentro de una cadena, con validación de errores léxicos.

Implementación de un parser para la evaluación de expresiones matemáticas dada una cadena del tipo char* cadena.

Diagrama del Autómata a utilizar

El siguiente diagrama es una representación del autómata que utilizamos.



Matriz de Transición

TT	+	-	0	1	3	4	5	6	7	8	9	a	b	c	d	e	f	A	B	C	D	E	F	x	X
0	2	2	1	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	5	5	5	5	5	5	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	3
2	-	-	2	2	2	2	2	2	2	2	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	-	-
4	-	-	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	-	-
5	-	-	5	5	5	5	5	5	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

A partir de esta matriz implementamos la función int transición(int estado, char c) que dado un estado y un carácter nos retorna el estado al que pasariamos después de consumir el carácter c.

```
int transición(int estado_actual, char c)
```

La implementación fue hecha con sentencias switch por cada carácter posible en caso de consumir algún carácter no válido la función retorna -1

Ejemplo:

transición(0, +) : Esto significa que estamos en el estado 0 (Estado inicial) y consumimos '+' Por lo tanto deberíamos pasar al estado 2 siguiendo la matriz.

También creamos una función llamada

```
void automata(char* cadena, int* pos_inicial, int* estado_actual)\
```

A la que le pasamos la cadena, una posición inicial desde donde va a analizar y el estado actual en el que nos encontramos.
La decisión de pasarle una posición inicial se debe a que como la cadena que obtenemos es toda una misma cadena donde cada Token está separado por '#' lo que hicimos fue armar una función auxiliar que cuenta la cantidad de caracteres que hay antes del siguiente '#' y retorna la cantidad

```
int cantHastaTok(char* cadena, int i){
    while(cadena[i] != '\0' && cadena[i]!='#'){
        i++;
    }
    return i;
}
```

Ejemplo:
cantHastaTok(123#9283#233, 0): Contaría desde el carácter con índice 0 hasta el primer '#' retornando 3 en este caso.

Con esta función pudimos recorrer toda la cadena sin necesidad de dividirla en Tokens. y como sabíamos exactamente los caracteres por Token siendo N el valor, llamábamos a la función autómata N veces, para que analice un Token completo y el último estado que retorne dependiendo de si es un estado de aceptación o no, determinaba si aceptamos o descartamos el Token.

Ejemplos de uso

Leyendo la cadena desde un archivo

```
mateo@DESKTOP-B7E9HII: /mnt/c/Users/Mateo/Desktop/test/repos-github/24-055-06/02-Automatas
mateo@DESKTOP-B7E9HII:/mnt/c/Users/Mateo/Desktop/test/repos-github/24-055-06/02-Automatas$ ./automata.out
Cadena leida del archivo: 123#075#0x1F#0xABCD#-42#12#07#-0x1#-077#0x#1234#0o7#0b101#-0xG#0x12#999#0o8#0123#0xABC#0b10#0

Leido: 123 -> Aceptado
Leido: 075 -> Aceptado
Leido: 0x1F -> Aceptado
Leido: 0xABCD -> Aceptado
Leido: -42 -> Aceptado
Leido: 12 -> Aceptado
Leido: 07 -> Aceptado
Leido: -0x1 -> No aceptado
Leido: -077 -> Aceptado
Leido: 0x -> No aceptado
Leido: 1234 -> Aceptado
Leido: 0o7 -> No aceptado
Leido: 0b101 -> No aceptado
Leido: -0xG -> No aceptado
Leido: 0x12 -> Aceptado
Leido: 999 -> Aceptado
Leido: 0o8 -> No aceptado
Leido: 0123 -> Aceptado
Leido: 0xABC -> Aceptado
Leido: 0b10 -> No aceptado
Leido: 0 -> Aceptado

Aceptados 14/21
6 decimales
4 octales
4 hexadecimales
```

Leyendo la cadena desde la terminal como argumento del Main() -> argv[1)

```
mateo@DESKTOP-B7E9HII: /mnt/c/Users/Mateo/Desktop/test/repos-github/24-055-06/02-Automatas
mateo@DESKTOP-B7E9HII:/mnt/c/Users/Mateo/Desktop/test/repos-github/24-055-06/02-Automatas$ ./automata.out 456#034#0x3B#0x1C4#
0x23#888#0o10#0775#0xD2#0b1010#0
Cadena leida de argv[1]: 456#034#0x3B#0x1C4#-98#064#017#-0x2#-055#0x3E7#246#0o5#0b110#-0xF#0x23#888#0o10#0775#0xD2#0b1010#0

Leido: 456 -> Aceptado
Leido: 034 -> Aceptado
Leido: 0x3B -> Aceptado
Leido: 0x1C4 -> Aceptado
Leido: -98 -> Aceptado
Leido: 064 -> Aceptado
Leido: 017 -> Aceptado
Leido: -0x2 -> No aceptado
Leido: -055 -> Aceptado
Leido: 0x3E7 -> Aceptado
Leido: 246 -> Aceptado
Leido: 0o5 -> No aceptado
Leido: 0b110 -> No aceptado
Leido: -0xF -> No aceptado
Leido: 0x23 -> Aceptado
Leido: 888 -> Aceptado
Leido: 0o10 -> No aceptado
Leido: 0775 -> Aceptado
Leido: 0xD2 -> Aceptado
Leido: 0b1010 -> No aceptado
Leido: 0 -> Aceptado

Aceptados 15/21
5 decimales
5 octales
5 hexadecimales
```

Parser

Para la implementación del parser nos guiamos del código fuente de un Descent Parser. Implementación detallada en el repositorio github de la cátedra.

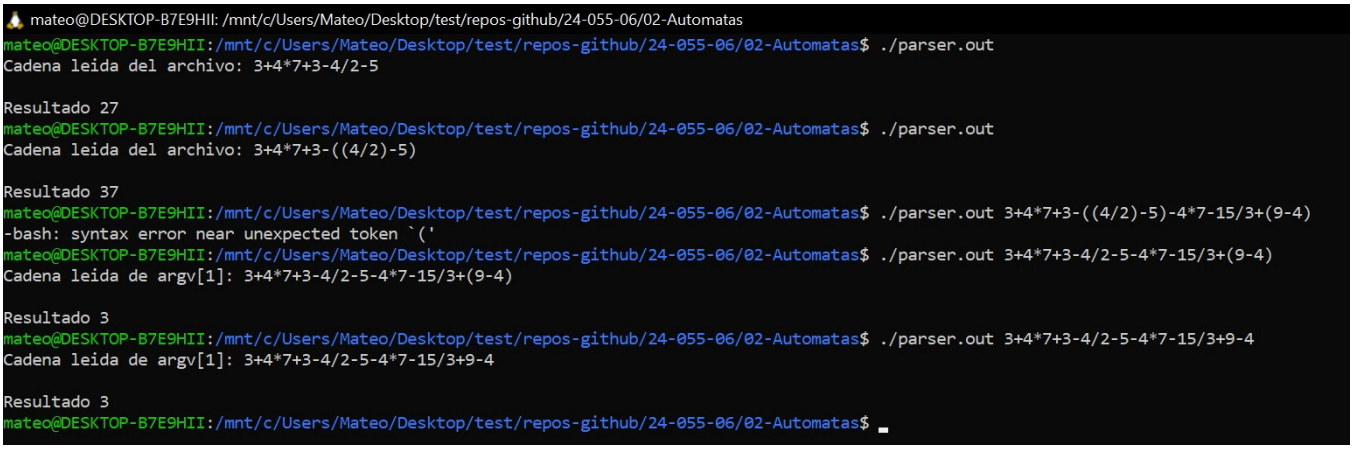
[Link Github] [Drive con los archivos]

Para el punto 2 usamos esta función:

```
int to_int(char c) {
    return c-'0'
}
```

Que dado un char numérico lo convertía en un int, esto fue útil para que el parser pueda evaluar la expresión ya que la cadena que le pasabamos era tipo char*

Ejemplos de uso



Fuentes:

- Evaluate char* expresions in C
  https://stackoverflow.com/questions/14472680/how-to-get-and-evaluate-the-expressions-from-a-string-in-c

- Top Down Parser
- https://www.youtube.com/watch?v=iddRD8tJi44

- Recursive Descent Parser

https://en.wikipedia.org/wiki/Recursive_descent_parser