INFORME TP AUTÓMATAS

Materia: SSL

**Explicación del pensamiento detrás del trabajo**

****

**Fecha de entrega:** 1/10/2023

**Link repositorio:** <https://github.com/GFCACACE/ssl-tp-automatas-g25>

**Integrantes Grupo 25:**

-CACACE, Guillermo Federico

-CALÓ, Ignacio

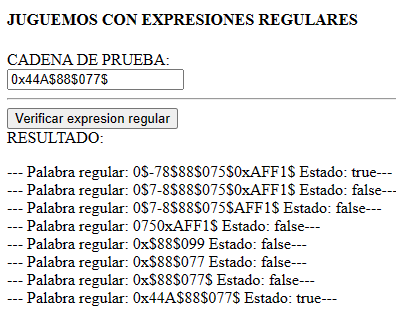
-GOMEZ PEREYRA, Manuel Francisco

- MAJER, Cecilia Alejandra

-TROSSERO, Agustín Francisco

***PUNTO 1***

Antes de comenzar con el diseño de los autómatas, primero dedujimos la ERX testeada en el html provisto en clase:



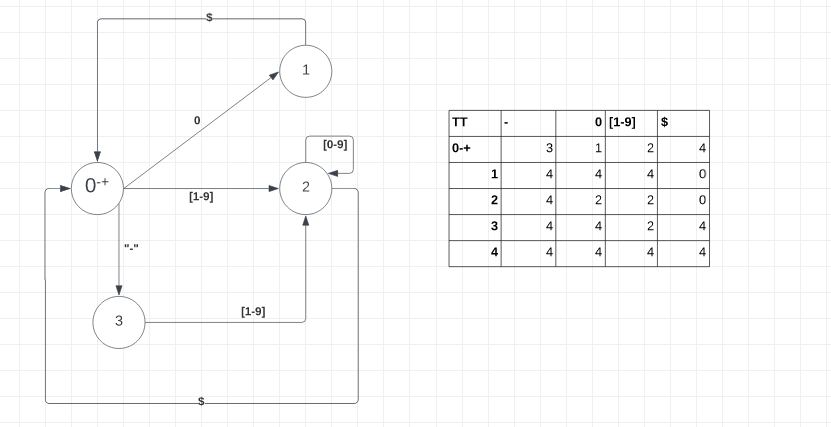
***ERX:***

/^((0[Xx][1-9A-Fa-f][0-9A-Fa-f]\*|0[1-7][0-7]\*|\-{0,1}[1-9][0-9]\*|0x0|0c0|0)\$)\*$/

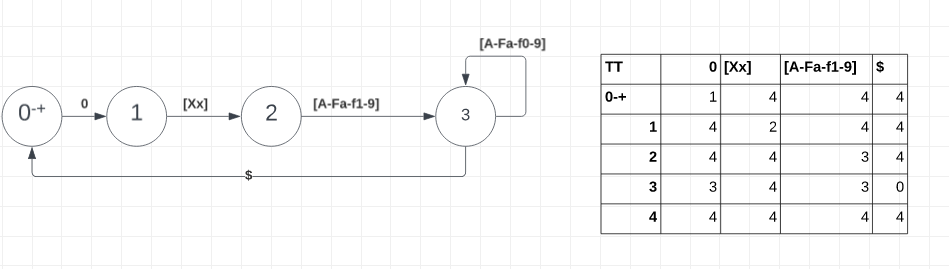
En el punto 1 decidimos hacer los tres autómatas por separado dado que consideramos que sería una forma más fácil de llevar a código, además de una forma más prolija de diagramar cada uno de los autómatas.

A continuación, los tres autómatas con sus respectivas TT:

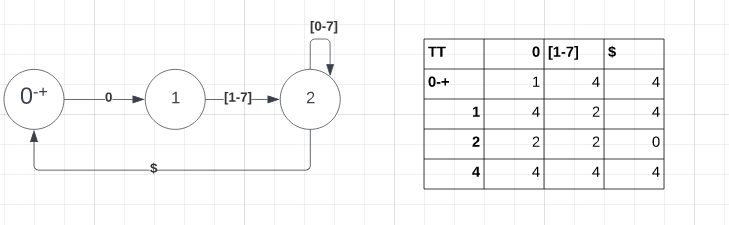
***Decimal:***



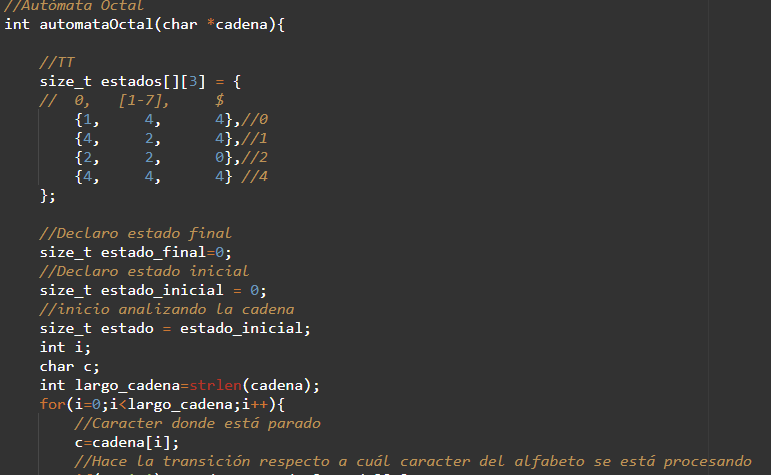
***Hexadecimal:***



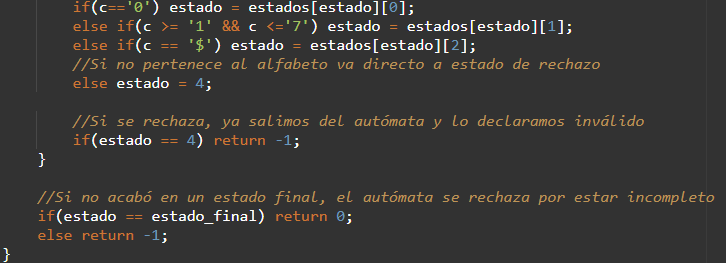
***Octal:***



Una vez hecho esto, pasamos a crearlos en C



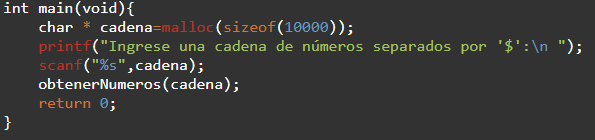
En estos se puede apreciar la tabla de transiciones hecha en una matriz llamada “estados”, los estados inicial y final (donde ambos son 0), y creamos una variable “estado” la cual marca la posición en donde estamos en la tabla. Por último obtenemos el largo de la cadena y pasamos a leerla caracter a caracter.



Esta parte varía según el autómata (hexadecimal, decimal y octal), pero la lógica es la misma. Cuando los if validan con algún carácter, se actualiza la fila a la que nos movemos en la tabla “estados” con la variable “estado”, en caso de que no reconozca algún carácter, nos movemos a la fila 4(rechazo), y al final comprobamos que el estado que salio del ciclo for sea un estado final(en este caso el 0), si es cierto, devolvemos 0, sino un -1.

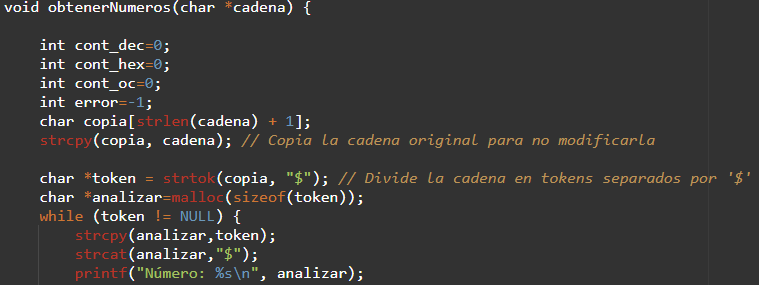
Luego juntaremos los tres autómatas para reconocer si en alguno de los tres hubo error léxico, en una función llamada “obtenerNumeros”

Primero declaramos una variable llamada “cadena” dándole un espacio específico, le pedimos al usuario que nos ingrese la cadena de números separada por $, y se la pasamos a la función que tiene los autómatas

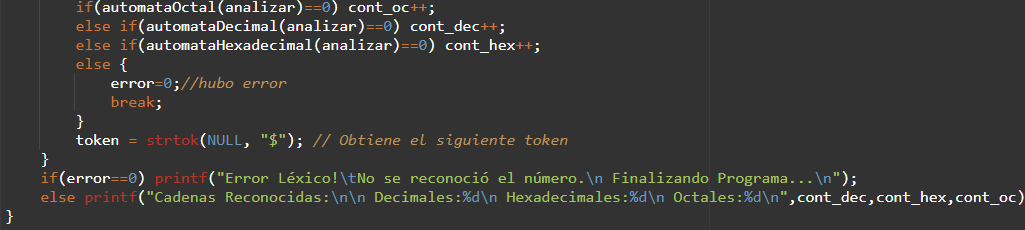


En esta función declaramos los contadores de cada autómata y copiamos la cadena que se recibe en una variable “copia” a la cual vamos dividiendo con “strtok” en partes que terminen con $, pero como esta función “agarra” todo lo que esté HASTA cierto carácter, luego de copiarlo en “analizar” tenemos que volverlo a agregar con “strcat”.

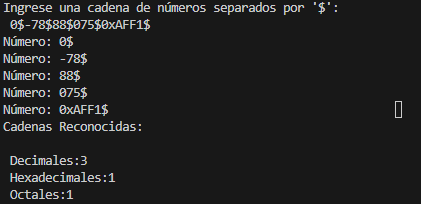
Mientras vaya reconociendo tokens, los va a ir mostrando en pantalla.



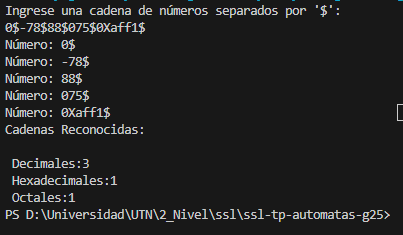
Dentro del while tenemos una cadena de if en donde la cadena obtenida se manda a cada autómata, y si uno de estos la reconoce, aumenta su contador. Una vez que sale de esta cadena, avanzamos a la siguiente parte de la expresión que termine con $.



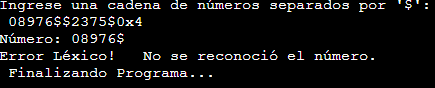
A continuación vemos un ejemplo de su funcionalidad, donde se pudo reconocer 3 cadenas decimales(-78, 88 y 0), una octal(075), y una hexadecimal(0xAFF1).



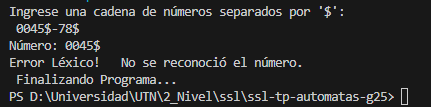
Abajo está el mismo ejemplo pero demostrando que admite tanto mayúsculas como minúsculas.



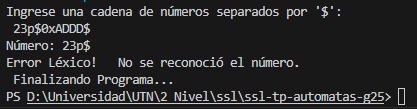
Un ejemplo de cómo rechaza cadenas incorrectas es el siguiente, ya que el autómata octal no admite 8 ni 9. El $$ es ignorado como se vé en el segundo.



Tampoco admite 0’s a la izquierda sea para cualquier sistema:

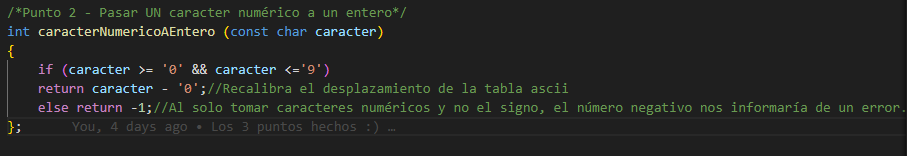


Otro caso de error, por ejemplo se ingresa una cadena que no es reconocida por ningún autómata porque hay un carácter ‘p’ colado en la cadena.



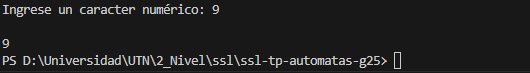
***PUNTO 2***

En el punto 2 (caracterNumericoAEntero) realizamos una función que dado un carácter numérico (del ‘0’ al ‘9’), traduzca mediante un cálculo a un entero operable teniendo en cuenta los valores de la tabla ASCII.

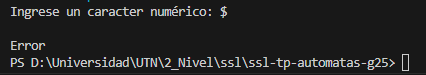


Tests:

1. Testeo número normal:



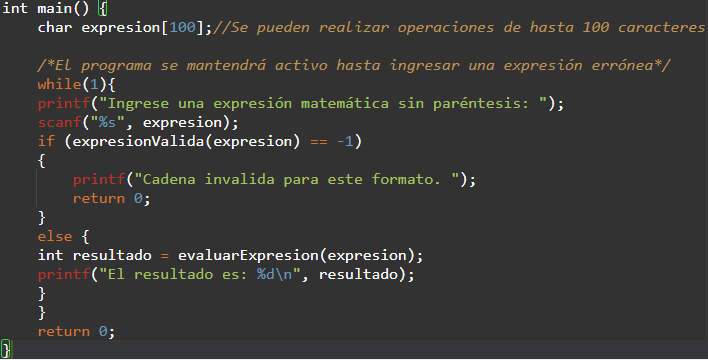
1. Testeo numero mal ingresado. Debe dar un error



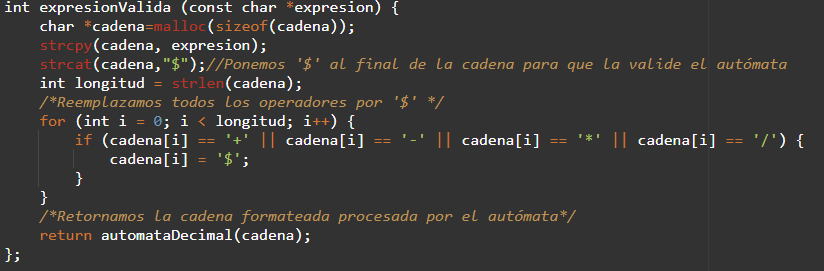
***PUNTO 3***

En el punto 3 decidimos optar por usar el autómata decimal, modificándolo brevemente en la función expresionValida (hacemos una copia de la expresión, en esa copia reemplazamos los operadores por “$”, y lo validamos con el autómata decimal), con la cual podemos verificar que la composición de la expresión matemática dada sea correcta.

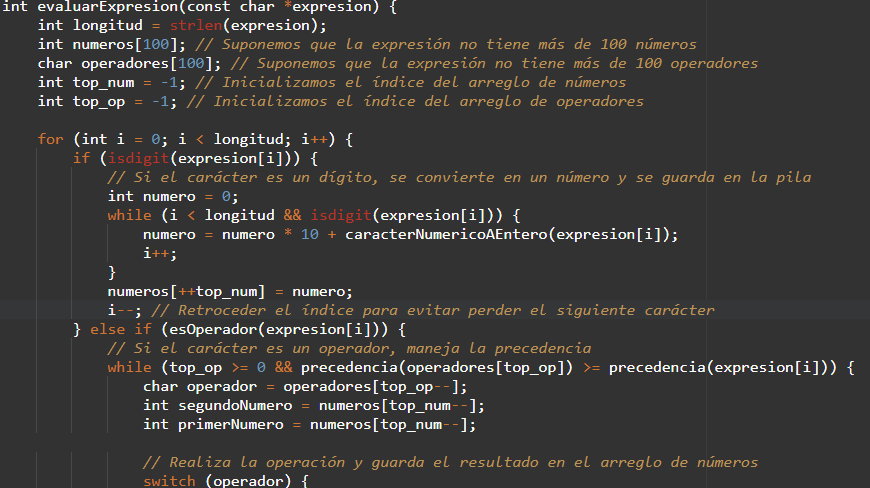
En la función principal pedimos que se ingrese una cadena que contenga una expresión matemática, y esta es validada por la función “expresiónValida”. En caso de pasar este control, se ejecutará “evaluarExpresión”.



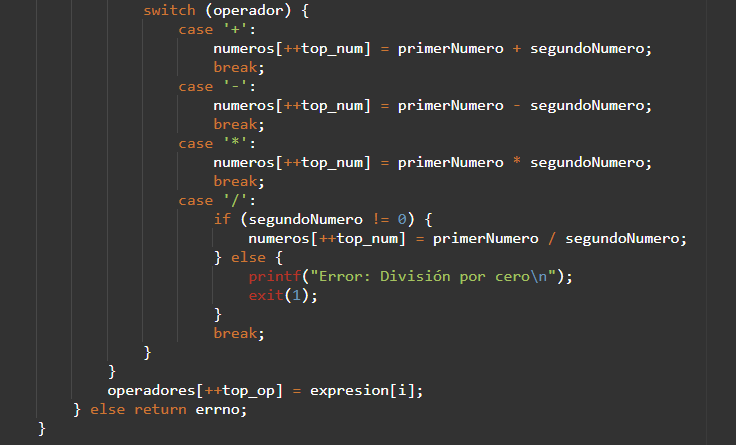
ExpresiónValida lee la cadena y reemplaza los símbolos de suma, resta, mul y div, por $, para luego pasarle esta cadena al automata decimal que hicimos en el punto 1



Luego de que la cadena está validada, se utiliza el algoritmo (evaluarExpresion) para resolver una cadena de chars, dividiéndolos entre dos arrays, una de “numeros” (utilizando un algoritmo similar al del punto 2) y otra de “operadores”.



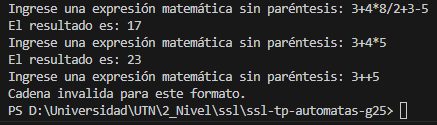
para luego establecer la precedencia con un switch, y hacemos lo mismo para realizar las operaciones

  
En el caso de que se ingrese una cadena inválida, se mostrará un mensaje de error.

Decidimos tomar como límite una operación matemática de hasta 100 caracteres para simplificar el código, ya que consideramos innecesaria y fútil la evaluación de una operación matemática simple más larga.  
Tests:

1. Testeo operación descrita en el TP (3+4\*8/2+3-5). Debe dar 17.
2. Otro testeo precedencia (Operación 3+4\*5). Debe dar 23.
3. Testeo cadena errónea (doble operador. Ej: 3++5). Debe dar error.

Debajo se muestran estos tres casos juntos:



1. Testeo caracter invalido (Ej: 3+4{\*5). Debe dar error:

