# G-ADUN


## Ejercicio 2: Expresiones regulares y operaciones con cadenas

## Ejercicio 2: Expresiones regulares y operaciones con cadenas

### Conceptos a trabajar:

- Operaciones con cadenas
- Expresiones Regulares
- Máquinas de estado finito

### Proyecto:

A través de GADUN, en la carpeta raíz de la herramienta busque en la sección de *"GRAMÁTICAS_EJERCICIOS"* el proyecto *"ejercicio2_lenguaje_básico*". Si no encuentra el directorio o el archivo por favor descargue los de Repositorio GADUN .

### Introducción:

Como se revisó en el anterior ejercicio en los analizadores léxicos los patrones son fundamentales en el momento de definir los conjuntos de elementos que conforman un lenguaje. Si bien GADUN permite la creación de Tokens de conjunto para reconocer expresiones específicas, en general los analizadores léxicos se basan únicamente en expresiones regulares para realizar el reconocimiento, de hecho los Tokens de conjunto de GADUN utilizan expresiones regulares en el fondo, es por eso que en este ejercicio podrá observar como la definición patrones a través de expresiones regulares debe ser un proceso bien ejecutado.

En este ejercicio trataremos de construir un analizador léxico que permita identificar expresiones básicas utilizadas en un lenguaje de programación.

| Conjuntos | Definición |
|---|---|
| Palabras reservadas | if , while , True , False |
| Tipos de variables | int , bool , char , string |
| Operadores | + , - , * , / , ^ |
| Números | 0, 1, 2, 3, 4…. N |
| Paréntesis | ( , ) |
| Separadores | ; , , |
| Variables | Secuencias alfanuméricas que empiezan por un caracter alfabetico |

*Tabla de conjuntos del analizador*

## Proceso

1. Abre el proyecto indicado para esta guía y dirígete a la sección de definición “Analizador Léxico”, para este ejercicio solo nos centraremos en esta área.

2. Como podrá observar el proyecto ya contiene en Lista de Tokens, los tokens ***reserved*** y ***types***, Tokens de conjunto que reconocen las expresiones para palabras reservadas y tipos de variables, respectivamente.

| | Nombre | Expresión |
|---|---|---|
| 1 | reserved | while,if,True,False |
| 2 | types | int,bool,char,string |

   Agregue a la lista de Tokens los Tokens de Conjunto que considere necesarios para reconocer los elementos de Operadores, Paréntesis y Separadores.

   Recuerde que utilizamos estos Tokens para reconocer expresiones específicas y de las que queremos tener control individual para el análisis sintáctico.

3. A continuación ubique en la sección para definición de Tokens la opción para definir tokens de Expresión Regular.

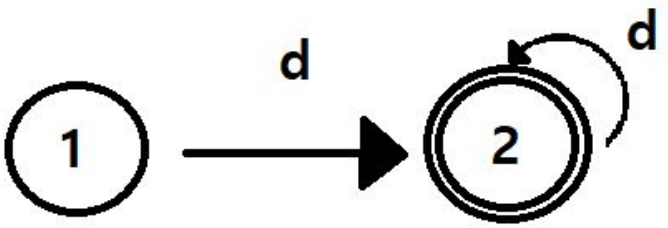

   El primer Token a definir será el Token necesario para reconocer números, ya que las expresiones numéricas son ilimitadas y no pueden ser definidas en un conjunto finito usamos una expresión Regular.

   La expresión regular para reconocer expresiones numéricas será: **d+**

   **d** (dígitos 0-9) y **+**(un carácter o más)

   Como verá la expresión consta de un solo conjunto de caracteres, el conjunto de dígitos y este está acompañado por un símbolo que indica la cantidad de elementos de ese conjunto en este caso **+** que indica la coincidencia de uno o más caracteres seguidos del mismo conjunto.

Si se hace un análisis profundo del funcionamiento de GADUN, encontrará que este utiliza la librería regex para el análisis Léxico, y al tiempo esta utiliza algoritmos semejantes a Máquinas de Estado Finito para realizar el reconocimiento.




Entonces GADUN lo que hace es utilizar estas MEF e ir ingresando caracteres hasta que una de estas máquinas reconozca la secuencia dando un estado válido.

En este caso para números indica a la máquina que el primer carácter deberá ser un dígito y el más expresa que si siguen llegando dígitos, la secuencia ingresada pertenece a ese patrón.

De esta manera GADUN reconoce conjuntos como:
d = dígitos
l = caracteres alfabéticos minúsculas
L = caracteres alfabéticos mayúsculas
**.** = cualquier carácter  a excepción de salto de línea

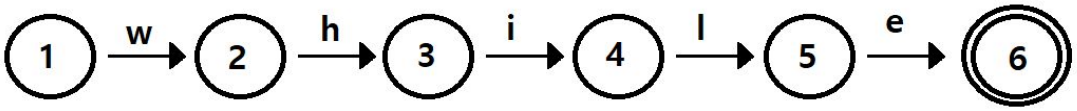
Pero también permite reconocer expresiones específicas, como las palabras reservadas por ejemplo, para reconocer **'while'** simplemente tendría que escribir en la expresión regular la palabra ***while*** ya que esta no contiene ningún carácter especial como los antes nombrados*(Para aprender más sobre caracteres especiales en REGEX consulte el sitio oficial)*.

4. Ahora definiremos el Token de expresión regular para reconocer variables, para esto deberemos entender que una variable es una secuencia de caracteres que empieza por una 'letra' y continua en una combinación de 'letras y números'.

   variable = letra + letras o números

   Antes de ver la expresión regular para variables en GADUN denote que en las expresiones regulares se pueden aplicar operaciones de cadenas, como por ejemplo la concatenación tome por por ejemplo la expresión ***while*** que sirve para reconocer la palabra **while.** En este caso lo que está indicando al

analizador en realidad es que busque una secuencia que empieza por **w**, continúe por **h** y así hasta la última letra, si pudiéramos ver la MEF para esto sería:




De la misma manera podemos hacer eso con los conjuntos por ejemplo:

**dl :** Un dígito seguido de una letra
**adl:** La letra a seguida de un dígito y una letra

De esta manera también se puede aplicar la operación de unión que indica el reconocimiento de varios conjuntos para un solo carácter, tome por ejemplo los Tokens de conjunto, estos tokens en realidad utilizan la operación de Unión para reconocer varias expresiones, así:

**(int | bool | char | string)** → Expresión regular utilizada en el token de conjunto Types

**(if | while | True | False)** → Expresión regular utilizada en el token de conjunto Reserved

**(d | l)** → Expresión que permite reconocer una letra o un número

De esta manera la expresión regular que reconocerá variables será:

**l (d | l) *** → Cualquier expresión que empieza con una letra minúscula y es seguida de letras o número

Observe que el símbolo más(+) y asterisco(*) permiten la aplicación de las cerradura positiva y cerradura de Kleene sobre el carácter inmediatamente anterior.

5. Por último observe cómo funcionan las potenciaciones dentro de las expresiones regulares de GADUN.

En algunos casos podría ser necesario el reconocimiento de un número específico de caracteres que cumplen con una misma regla, por ejemplo si se quiere reconocer palabras con solamente cuatro caracteres. para esto podría simplemente reutilizarse la expresión para reconocer un caracter, asi:

llll → Reconoce una secuencia de cuatro caracteres

Que fácilmente permite reconocer las expresiones esperadas, pero esto podría no ser muy práctico a la hora de reconocer expresiones con valores más altos y más diversos, tome por ejemplo, si se quiere reconocer una secuencia que comience por 5 letras seguidas de 4 números y finalice de dos a cuatro letras.

lllllddddd(ll)* → 5 letras, 4 números y de 0 a 2 letras

para estos casos Regex permite la creación de potencias utilizando números y los símbolos {}, de esta manera:

l{4} → Reconoce una secuencia de cuatro caracteres

l{5}d{4}l{0-2} → 5 letras, 4 números y de 0 a 2 letras

Creando de esta manera expresiones equivalentes que pueden ser especificadas de una mejor manera.

## Ejercicio de refuerzo:

1. Construye el diagrama de la máquina de estado finito para la expresión regular de variables.

2. Crea un Token de expresión regular que permita reconocer operadores a través de una expresión regular y observa en qué se diferencia su funcionamiento con un token de tipo conjunto

3. Consulta cómo definir una expresión regular que permita reconocer cualquier secuencia de caracteres que se encuentre entre **“”**