

**DISEÑO E IMPLEMENTACION DE LEXER Y PARSER**

**Resistencia, Chaco**

**2022**

**3 DE JULIO**

**ALUMNOS**

-Arrejin, Sixto Feliciano

-Maciel Meister, Tobias Alejandro

-San Lorenzo, André Leandro

**PRIMER CUATRIMESTRE**

I.S.I

Universidad Tecnológica Nacional FRRe

**SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES**

**ÍNDICE**

[**INTRODUCCIÓN** 1](#_Toc107784312)

[**RSS: CARACTERÍSTICAS** 1](#_Toc107784313)

[**Componentes léxicos y etiquetas** 1](#_Toc107784314)

[**GRAMÁTICA** 3](#_Toc107784315)

[**Símbolos de la gramática** 3](#_Toc107784316)

[**Reglas de producción de la gramática** 3](#_Toc107784317)

[**ANÁLISIS LÉXICO** 8](#_Toc107784318)

[**ANÁLISIS SINTACTICO** 9](#_Toc107784319)

[**Funciones auxiliares** 10](#_Toc107784320)

[**Modo de obtención del interprete** 12](#_Toc107784321)

[**Modo de ejecución del interprete** 12](#_Toc107784322)

[**Interactiva** 12](#_Toc107784323)

[**A partir de un archivo** 13](#_Toc107784324)

[**Ejemplos** 15](#_Toc107784325)

[**CONCLUSIÓN** 18](#_Toc107784326)

[**Reflexión sobre el trabajo realizado** 18](#_Toc107784327)

[**Problemas en el desarrollo del interprete** 19](#_Toc107784328)

[**Puntos fuertes y débiles del intérprete desarrollado** 19](#_Toc107784329)

[**RECURSOS UTILIZADOS** 20](#_Toc107784330)

# **INTRODUCCIÓN**

En el presente trabajo se describe el desarrollo y puesta en marcha de un analizador léxico y sintáctico que permitirá analizar, validar y transformar un archivo en formato RSS.

En rasgos generales, este analizador deberá recibir un archivo en formato RSS e indicar si este cumple con las reglas de un archivo RSS (es decir, si esta adecuado al estándar, sin errores), por el contrario, deberá indicar en donde se encuentra el error. Finalmente deberá convertir el contenido del archivo RSS en un documento HTML válido.

Para poder llevar a cabo el trabajo, primero que nada, debemos entender que es un RSS y que características posee.

RSS es una forma de estructurar datos en XML, el cual es muy útil para la divulgación de información de canales o páginas webs debido a la facilidad que presenta al mantenerse actualizado en tiempo real. Es decir, que los lectores pueden mantenerse informados sobre webs o noticias de su interés, almacenando todo en un solo lugar, lo que les permite ahorrar tiempo.

Ya que RSS se sustenta en XML es indispensable entender de qué se trata este lenguaje. XML es un lenguaje utilizado para el almacenamiento de datos de manera legible, su principal aplicación la vemos en las páginas web gracias a lo simple que resulta el manejo de datos en el mismo.

# **RSS: CARACTERÍSTICAS**

## **Componentes léxicos y etiquetas**

A continuación, se encuentran las etiquetas de RSS y estructuras que se incluirán en el trabajo. Existen más etiquetas opcionales, pero no se tendrán en cuenta todas para el desarrollo del proyecto.

**Versión de XML**

* En la primera línea del código se indica la versión XML, que puede contener la codificación a utilizarse. Esta etiqueta es la única que no contienen cierre. Además, aparece una sola vez.

**Versión de RSS**

* En la segunda línea se realiza la declaración del formato RSS dentro del documento, especificando la versión que se utilizará.
* Posee una etiqueta de cierre.
* Entre la declaración y la etiqueta del cierre se encuentran todos los elementos que se mencionaran a continuación.

**Channel**

* Posee una etiqueta de apertura y una de cierre. Aparece una sola vez en el documento. Dentro de esta etiqueta poseemos los siguientes elementos obligatorios los cuales pueden aparecer en cualquier orden (aunque solo consideraremos el siguiente orden):
  + **<title>:** establece el título del canal y está constituido por cadena de texto.
  + **<link>:** define el enlace al canal y posee una URL.
  + **<description>:** brinda una descripción del canal y está formada por cadena de texto.
  + **<item>:** es el único elemento que puede aparecer más de una vez. Posee los siguientes elementos secundarios obligatorios:
    - **<title>**: establece el título del artículo y está constituido por cadena de texto.
    - **<link>**: define el enlace al artículo.
    - **<description>**: brinda una descripción del artículo y está formada por cadena de texto

Esta etiqueta posee múltiples elementos secundarios opcionales, no obstante, solo utilizaremos el siguiente:

* + - **<category>**: define una categoría para el agrupamiento de los distintos artículos. Formado por cadena de texto.
* Los elementos opcionales secundarios que se encuentran dentro de <channel> pueden aparecer también en diferente orden. A diferencia de los anteriormente mencionados, estos pueden no aparecer. Los que consideramos son los siguientes:
  + **<category>:** define una categoría para el agrupamiento de diferentes páginas. Formado por cadena de texto.
  + **<copyright>:** brinda información acerca de los derechos o restricciones. Formado por cadena de texto.
  + **<image>:** solo se puede definir una imagen por canal. Cuenta con las siguientes subelementos obligatorios:
    - **<url>:** es la dirección concreta de la imagen. Tiene la siguiente estructura:

protocolo://dominio:puerto/ruta#fragmento

* + - **<title>:** establece un título para la sección (para la imagen).
    - **<link>:** define el enlace a la imagen y está constituido por una URL.

La imagen cuenta con diferentes elementos opcionales. Nosotros tendremos en cuenta solamente dos, los cuales son:

* **<height>:** muestra la altura de la imagen.
* **<width>:** indica el ancho de la imagen.

Estos dos últimos elementos están constituidos por números.

# **GRAMÁTICA**

## **Símbolos de la gramática**

Los símbolos terminales y no terminales que se utilizaran en la gramática son los siguientes:

**Símbolo sentencia** =

**Terminales** = {TEXT, NUMERO, CHANNEL1, CHANNEL2, ITEM1, ITEM2, ENCODING, TITLE1, TITLE2, LINK1, LINK2, URL1, URL2, IMAGE2, DESCRIPTION1, DESCRIPTION2, CATEGORY1, CATEGORY2, COPYRIGHT1, COPYRIGHT2, IMAGE1, WIDTH1, WIDTH2, HEIGHT1, HEIGHT2, RSS2, RSS1, XML, TOKENURL, VERSION1, VERSION2}

**No terminales**= {main, cuerpo, tn, item, copyright, image, wh}

Es importante destacar que los terminales serán tratados como tokens ya que serán declaradas expresiones regulares para los mismos en el desarrollo del lexer.

## **Reglas de producción de la gramática**

A continuación, se encuentran las reglas de producción con una breve descripción de las mismas para comprender mejor el funcionamiento de la gramática.

primera derivación que se realiza, en la cual nos permitirá derivar en <main> <cuerpo>

🡪 XML VERSION1 ENCODING RSS1 VERSION2 CHANNEL1 **<main> <cuerpo>** CHANNEL2 RSS2

🡪 XML VERSION2 RSS1 VERSION2 CHANNEL1 **<main> <cuerpo>** CHANNEL2 RSS

main 🡪 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2

main 🡪 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 CATEGORY1 **<tn>** CATEGORY2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2 CATEGORY1 **<tn>** CATEGORY2 LINK1 TOKENURL LINK2

main 🡪 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2 CATEGORY1 **<tn>** CATEGORY2

main 🡪 TITLE1 **<tn>** TITLE2 CATEGORY1 **<tn>** CATEGORY2 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 TITLE1 **<tn>** TITLE2 CATEGORY1 **<tn>** CATEGORY2 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2

main 🡪 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2 CATEGORY1 **<tn>** CATEGORY2

main 🡪 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2

main 🡪 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2 CATEGORY1 **<tn>** CATEGORY2

main 🡪 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 CATEGORY1 **<tn>** CATEGORY2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2 CATEGORY1 **<tn>** CATEGORY2

main 🡪 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2 CATEGORY1 **<tn>** CATEGORY2 TITLE1 **<tn>** TITLE2

main 🡪 LINK1 TOKENURL LINK2 CATEGORY1 **<tn>** CATEGORY2 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 LINK1 TOKENURL LINK2 CATEGORY1 **<tn>** CATEGORY2 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2

El main permite realizar todas las combinaciones posibles que encabezan el canal y son compartidas con los items

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 CATEGORY1 **<tn>** CATEGORY2

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2 CATEGORY1 **<tn>** CATEGORY2 LINK1 TOKENURL LINK2

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 CATEGORY1 **<tn>** CATEGORY2

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2 CATEGORY1 **<tn>** CATEGORY2 TITLE1 **<tn>** TITLE2

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 CATEGORY1 **<tn>** CATEGORY2 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2

main 🡪 DESCRIPTION1 **<tn>** DESCRIPTION2 CATEGORY1 **<tn>** CATEGORY2 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2

main 🡪 CATEGORY1 **<tn>** CATEGORY2 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 CATEGORY1 **<tn>** CATEGORY2 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2

main 🡪 CATEGORY1 **<tn>** CATEGORY2 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 DESCRIPTION1 **<tn>** DESCRIPTION2

main 🡪 CATEGORY1 **<tn>** CATEGORY2 LINK1 TOKENURL LINK2 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2

main 🡪 CATEGORY1 **<tn>** CATEGORY2 DESCRIPTION1 **<tn>** DESCRIPTION2 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2

main 🡪 CATEGORY1 **<tn>** CATEGORY2 DESCRIPTION1 **<tn>** DESCRIPTION2 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2

tn permite derivar a texto y textos antecedidos por números

tn 🡪 NUMERO TEXT

tn 🡪 TEXT

cuerpo 🡪 **<item>**

cuerpo 🡪 **<copyright> <item>**

cuerpo 🡪 **<image> <item>**

cuerpo 🡪 **<item> <image>**

Cuerpo principal que permite la creación de ítems con sus distintas combinaciones

cuerpo 🡪 **<copyright> <image> <item>**

cuerpo 🡪 **<copyright> <item> <image>**

cuerpo 🡪 **<image> <copyright> <item>**

cuerpo 🡪 **<image> <item> <copyright>**

cuerpo 🡪 **<item> <image> <copyright>**

cuerpo 🡪 **<item> <copyright> <image>**

Producción ítem, la cual deja generar uno solo o los deseados

item 🡪 ITEM1 **<main>** ITEM2

item 🡪 ITEM1 **<main>** ITEM2 **<item>**

Producción para etiqueta copyright

copyright 🡪 COPYRIGHT1 **<tn>** COPYRIGHT2

image 🡪 IMAGE1 URL1 TOKENURL URL2 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 **<wh>** IMAGE2

image 🡪 IMAGE1 URL1 TOKENURL URL2 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 **<wh>** IMAGE2

image 🡪 IMAGE1 TITLE1 **<tn>** TITLE2 URL1 TOKENURL URL2 LINK1 TOKENURL LINK2 **<wh>** IMAGE2

image 🡪 IMAGE1 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 URL1 TOKENURL URL2 **<wh>** IMAGE2

image 🡪 IMAGE1 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 URL1 TOKENURL URL2 **<wh>** IMAGE2

image 🡪 IMAGE1 LINK1 TOKENURL LINK2 URL1 TOKENURL URL2 TITLE1 **<tn>** TITLE2 **<wh>** IMAGE2

image 🡪 IMAGE1 URL1 TOKENURL URL2 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 IMAGE2

image 🡪 IMAGE1 URL1 TOKENURL URL2 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 IMAGE2

image 🡪 IMAGE1 TITLE1 **<tn>** TITLE2 URL1 TOKENURL URL2 LINK1 TOKENURL LINK2 IMAGE2

image 🡪 IMAGE1 TITLE1 **<tn>** TITLE2 LINK1 TOKENURL LINK2 URL1 TOKENURL URL2 IMAGE2

image 🡪 IMAGE1 LINK1 TOKENURL LINK2 TITLE1 **<tn>** TITLE2 URL1 TOKENURL URL2 IMAGE2

image 🡪 IMAGE1 LINK1 TOKENURL LINK2 URL1 TOKENURL URL2 TITLE1 **<tn>** TITLE2 IMAGE2

Producción que permite todas las combinaciones de altura y ancho de imágenes

wh 🡪 HEIGHT1 NUMERO HEIGHT2

wh 🡪 WIDTH1 NUMERO WIDTH2

wh 🡪 HEIGHT1 NUMERO HEIGHT2 WIDTH1 NUMERO WIDTH2

wh 🡪 WIDTH1 NUMERO WIDTH2 HEIGHT1 NUMERO HEIGHT2

Antes de comenzar a explicar en qué consiste el código del lexer es importante hablar acerca de la herramienta empleada para el desarrollo del mismo. Luego de un análisis de las diferentes opciones se consideró que lo más optimo era utilizar ply (Python Lex and Yacc). Esta herramienta esta escrita exclusivamente en Python y es básicamente una implementación de Lex y Yacc los cuales están escritos en C.

Una de las ventajas más importantes de ply es que nos permite desarrollar en un mismo lenguaje de programación tanto el lexer como el parser, lo cual facilita el desarrollo del analizador léxico sintáctico en comparación a si tuviéramos que realizarlos de manera separada cada uno para luego complementarlos.

Por otra parte, Python es un lenguaje de programación en el cual la mayoría de los integrantes del grupo ya había tenido contacto. Además de que vimos una oportunidad de fortalecer los conocimientos en el mismo.

# **ANÁLISIS LÉXICO**

A continuación, se presentará una breve descripción del archivo del lexer.

En primer lugar, fue necesario el uso algunas librerías, como re, para el manejo de las expresiones regulares en Python además de la propia librería de ply para la implementación del analizador léxico. También se utilizaron algunas otras librerías que facilitaron el manejo de algunas cuestiones concernientes al desarrollo del código.

import ply.lex as lex

import codecs

import os

import re

import sys

Posteriormente se encuentra la definición de todos los tokens con los que trabajaremos. Esto lo realizamos definiéndolos a todos como elementos de una misma lista denominada tokens. Como se puede observar, entre ellos se encuentran la mayoría de los tokens que definimos con anterioridad en la gramática, sin embargo, hay casos en los que unificamos algunos tokens y otros en los que se eliminaron.

tokens = [ "TEXT" ,"NUMERO",

        "CHANNEL1", "CHANNEL2",

        "ITEM1", "ITEM2","ENCODING",

        "TITLE1", "TITLE2",

        "LINK1", "LINK2",

        "URL1", "URL2", "IMAGE2",

        "DESCRIPTION1", "DESCRIPTION2",

        "CATEGORY1", "CATEGORY2",

        "COPYRIGHT1", "COPYRIGHT2",

        "IMAGE1", "WIDTH1","WIDTH2",

        "HEIGHT1","HEIGHT2",

        "RSS2","RSS1", "XML","TOKENURL",

        "VERSION1","VERSION2",

        ]

Por la manera en la que está estructurado ply, para definir las expresiones regulares de los tokens existen dos maneras de hacerlo, una simple y una compleja. Nosotros optamos por la forma compleja por la propia característica que tenían nuestros tokens; de esta manera, la expresión regular de cada token se establece dentro de una “función” con otras líneas de código que se ejecutaran al momento de encontrar dicho token.

def t\_URL1(t):

    r'<url>'

    return t

def t\_URL2(t):

    r'</url>'

    return t

De manera simple, definimos una expresión regular para aquellos caracteres en los que no requerimos de ninguna acción extra, simplemente son ignorados.

t\_ignore  = '\r'

También se definió una expresión regular especial en el caso de que no se reconozcan tokens definidos previamente, este detiene el análisis léxico

def t\_error(t):

    t.lexer.skip(1)

Finalmente se realiza la declaración del analizador léxico

lexer = lex.lex()

# **ANÁLISIS SINTACTICO**

A continuación, se encuentra una breve descripción del archivo referente al parser.

En primera instancia, para llevar a cabo el analizador sintáctico se requirió de la librería referente a la misma

import ply.yacc as yacc

A continuación, se realizaron las declaraciones de todas las producciones pertenecientes a la gramática definida.

En las producciones, las palabras en mayúsculas representan a los terminales, siendo estos mismos los tokens pasados del lexer al parser, mientras que las palabras minúsculas representan los no terminales, cada uno de los cuales tiene su propia producción.

def p\_sigma(p):

    '''sigma : XML VERSION1 ENCODING RSS1 VERSION2 CHANNEL1 main cuerpo CHANNEL2 RSS2

        | XML VERSION2 RSS1 VERSION2 CHANNEL1 main cuerpo CHANNEL2 RSS2'''

Luego de declarar todas las producciones definidas se define una producción adicional que se encargara de mostrar un error por pantalla en el caso de que haya un problema sintáctico

def p\_error(p):

    global correcto

    correcto=1

    pantalla1.insert('1.0', "Error de sintaxis en la línea "+str(p.lineno)+"\nVerifique la cadena "+str(p.value)+"\n\n")

Finalmente se realiza la declaración del analizador sintáctico

parser = yacc.yacc()

## **Funciones auxiliares**

Luego de la codificación del lexer y el parser se encuentra finalmente las líneas de código referentes a la interfaz gráfica utilizada por el intérprete.

En primera instancia se requirió de la librería tkinter que es una de las maneras más básicas que se puede encontrar en Python de implementar una interfaz. De esta librería se exportaron numerosas funcionalidades como se puede apreciar a continuación:

from tkinter import Entry, Frame, Label, Tk,Text, filedialog, ttk, messagebox

En primer lugar, se realizó la declaración de una ventana y su configuración:

ventana\_principal=Tk()

ventana=Frame(ventana\_principal)

ventana\_principal.config(bg="thistle4")

A continuación, se codificó las funciones referentes a los botones que se mostrarían en pantalla, así como también del mensaje que se emite como respuesta a una correcta compilación:

def copiar():

    global codigo, file, correcto

    codigo=pantalla.get("1.0",'end-1c')

    file = open(NombreArch.replace(".rss",".html"), "w")

    lexer.lineno =1

    correcto=0

    parser.parse(codigo)

    file.close()

    if codigo!="":

        if correcto==0:

            messagebox.showinfo(":)", "Compilación Exitosa")

#Funcion abrir archivo

def abrirarchivo():

    global NombreArch

    NombreArch=filedialog.askopenfilename(initialdir = "/Archivosprueba", title = "Select a File", filetypes = (("Rss files", "\*.rss\*"), ("Text files", "\*.txt\*"), ("all files", "\*.\*")))

    Archivo= open(NombreArch, "r", encoding="utf-8")

    pantalla.delete("1.0","end")

    pantalla.insert('1.0', Archivo.read())

Posteriormente se encuentra la codificación de todas las cajas de textos que se mostraran en la ventana, títulos y la creación y configuración de los botones:

#Texto

Label\_text= Entry(ventana)

Label\_text= Label(text= "LEXER Y PARSER")

Label\_text.config(font=('Helvetica 25 underline bold'), background="thistle4", foreground="gray10")

Label\_text.grid(row=0, column=0, columnspan=5, pady=20)

#Botones

style=ttk.Style()

style.configure('W.TButton', font =('calibri', 10, 'bold', 'underline'),border=5, borderwidth=5, background='#ab28bd', bd=5)

#Boton Abrir archivo

ventana.boton\_abrirarchivo= ttk.Button(text="Abrir Archivo", command=abrirarchivo, style='W.TButton',width=20).grid(row=3,column=0,pady=15,columnspan=2)

Finalmente se encuentra el llamado de la ventana, esto permite que se muestre la ventana por pantalla

ventana\_principal.mainloop()

## **Modo de obtención del interprete**

Para ejecutar el intérprete simplemente se debe dar doble click sobre el archivo “Interprete.exe” que se encuentra en la carpeta “directorio bin”. Este ejecutable se obtuvo directamente del código fuente correspondiente al interprete dentro del cual ya se encontraba tanto el analizador léxico como el sintáctico.

Obligatoriamente, no se requiere de ningún archivo adicional para ejecutar el intérprete, sin embargo, al ejecutarse este puede requerir de archivos del tipo “.rss” en el caso de que se desee abrir un archivo de este tipo desde algún directorio del pc para su análisis, estos pueden abrirse sin importar su nombre siempre y cuando cuenten con la extensión “.rss”.

## **Modo de ejecución del interprete**

El intérprete desarrollado tiene la posibilidad de funcionar de dos maneras diferentes. Por un lado, el usuario puede ingresar código a mano para analizarlo, por otro lado, también cuenta con la posibilidad de abrir y analizar un archivo “.rss” en concreto que se encuentre en cualquier directorio del pc.

### **Interactiva**

Al ejecutar el programa es posible simplemente empezar a escribir código en RSS directamente en el cuadro de texto que se encuentra en la parte superior del programa por medio del teclado de la siguiente manera:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Al finalizar es posible dar click al botón en la parte inferior que dice "Compilar” para analizar lo escrito, en el caso de que la compilación sea exitosa, se creara, en la carpeta donde se encuentre en ejecutable, el archivo “.html” correspondiente con un nombre por defecto “ArchivoConvertido”.

### **A partir de un archivo**

Al ejecutar el programa se cuentan con 3 botones en la parte inferior, se debe dar click al que dice “Abrir archivo”:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

De inmediato se abrirá el seleccionador de archivos. Por defecto este se posicionará en la carpeta donde se encuentran los archivos de prueba a utilizar con el intérprete, sin embargo, se puede navegar entre los directorios del pc y elegir cualquier otro:

Captura de pantalla de computadora

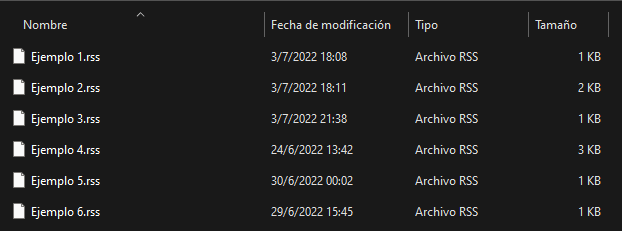
Descripción generada automáticamente

Una vez elegido el archivo este se abrirá en el cuadro de texto de la parte superior del interprete. Es posible compilarlo al dar click en el botón de “Compilar”, o continuar modificándolo escribiendo por teclado para luego compilarlo.

Al realizar la compilación, si este fue exitoso, se creará un documento “.html” en la misma carpeta desde donde abrimos el archivo, este poseerá el mismo nombre que el documento “.rss” abierto.

## **Ejemplos**

En los ejemplos a utilizar se encuentran los dos ejemplos propuestos por los docentes y cuatro ejemplos propuestos por los integrantes del grupo. Estos se encuentran en la carpeta “prueba” y son los siguientes:



A continuación, se puede ver la ejecución y resultado de analizar el archivo “Ejemplo 1.rss” con el intérprete creado.

El ejemplo 1 posee lo siguiente:

<?xml version="1.0" encoding="UTF-8" ?>

<rss version="2.0">

<channel>

    <title>RSS de la catedra de Sintaxis y Semantica de Lenguajes </title>

    <link>https://frre.cvg.utn.edu.ar/course/view.php?id399</link>

    <description>Sintaxis y Semantica de Lenguajes de la U.T.N. F.R.Resistencia. </description>

    <item>

        <title>Planificacion 2022</title>

        <link>https://frre.cvg.utn.edu.ar</link>

        <description>Planificacion de catedra, con cronograma de clases y evaluaciones</description>

    </item>

    <item>

        <title>Guía de Trabajos prácticos</title>

        <link>https://frre.cvg.utn.edu.ar/mod/resource/view.php?id=43544</link>

        <description>Guía de ejercicios propuestos a resolver en clase práctica</description>

    </item>

    <item>

        <title>Enunciado TPI</title>

        <link>https://wl</link>

        <description>Enunciado del Trabajo practico integrador</description>

    </item>

</channel>

</rss>

Abrimos el interprete y seleccionamos el archivo correspondiente:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Damos click en botón central “Compilar” y observamos el siguiente mensaje que nos corrobora que todo esta bien escrito:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Luego de esto podemos ver en la carpeta donde se encuentra el ejemplo seleccionado, que sea generado el archivo “.html” correspondiente al archivo analizado.

Imagen que contiene Texto

Descripción generada automáticamente

Si le abrimos este archivo en el navegador podemos comprobar que muestra los títulos y descripciones correspondientes.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

# **CONCLUSIÓN**

### **Reflexión sobre el trabajo realizado**

Para concluir el trabajo vamos a dar una valoración general de la experiencia que tuvimos en el desarrollo del mismo y de los resultados obtenidos.

Consideramos que pudimos confeccionar un analizar léxico sintáctico que cumple con los requerimientos propuestos. Además de ellos se implementaron todas las funcionalidades del mismo a través de una interfaz gráfica que permitió que el mismo sea usado más fácilmente. Así, cualquier persona puede utilizar el programa sin prácticamente ningún conocimiento.

Por otra parte, se destaca que este trabajo nos permitió conocer más sobre los compiladores y su funcionamiento. Constantemente estamos usando los mismos sin ponernos a pensar la forma en que estos actúan y todo el conocimiento teórico que está involucrado detrás de su funcionamiento.

En definitiva, desde nuestro punto de vista, lo más destacable fue la posibilidad de emplear todos los conocimientos vistos a lo largo del desarrollo de la materia en un proyecto practico. En muchas ocasiones, el nivel de abstracción de la materia es muy elevado y parece no tener ninguna relación con algo de la vida real. No obstante, al desarrollar el analizador pudimos ver como todos estos fundamentados en realidad están desarrollados con cierto propósito. Además, para realizar el proyecto fueron necesarios prácticamente todos los temas vistos en la materia, del primer al último día de clases. Es así que resultó altamente satisfactorio ver como todo tuvo un cierre en el TPI.

### **Problemas en el desarrollo del interprete**

Para cada una de las etapas fue necesario lo desarrollado en la etapa anterior, lo cual provocaba que se acarren algunos problemas entre etapas. Muchas veces lo que parecía ideal en una etapa, para la implementación posterior generaba ciertos conflictos con los cuales no se contaba, esto nos obligo dar pasos hacia atrás para poder avanzar con el desarrollo, de tal manera, prácticamente todo lo que habíamos desarrollado en un principio en cierto punto del trabajo se vio modificado.

Tuvimos especialmente inconvenientes durante el desarrollo final del parser, ya que al usar las producciones y los tokens provenientes del lexer, las derivaciones no se realizaban de una manera satisfactoria. Esto nos obligo a replantear tanto los tokens como las producciones, para así darle al interprete la menor cantidad de posibilidades posibles y caminos de derivación más claros. Esto implico que prácticamente reescribiéramos el lexer, para que tokenice los archivos de manera más eficiente para su análisis.

Por otro lado, también se nos presentaron algunos problemas a la hora de codificar la conversión a archivos HTML, ya que había etiquetas que algunas ocasiones debían convertirse y otras no. Sin embargo, esto fue aún más fácil de solucionar. El control de ello lo llevamos a cabo mediante la implementación de banderas en varias instancias del lexer.

### **Puntos fuertes y débiles del intérprete desarrollado**

Puntos fuertes:

* Posee una interfaz gráfica que la hace bastante amigable, es decir que es fácil de utilizar.
* El interprete es capaz de reconocer todas las combinaciones de etiquetas propuestas por la catedra.
* El código fuente se encuentra bien estructurado y organizado, lo que permite agregar modificaciones e interpretarlo de manera sencilla.

Puntos débiles:

* El interprete no indica los números de líneas del cuadro de texto que se ve en pantalla.
* No es tan flexible a cambios debido a la forma en la que fueron definidos los tokens.
* Para el caso de que se escriba código por teclado el archivo “.html” generado por cada compilación siempre tiene el mismo nombre, por lo que se reescribe por cada compilación.

# **RECURSOS UTILIZADOS**

* <http://www.w3big.com/es/rss/rss-syntax.html>
* <https://metricool.com/es/que-es-feed-rss/>
* <https://validator.w3.org/feed/check.cgi>
* <https://servicios.elpais.com/rss/>