Instituto Politécnico Nacional
Escuela Superior de Cómputo

Trabajo terminal

“Procesador de texto amigable con el usuario”
16-2-006

Presenta:
Montalvo Lezama Mario

Director:

Saucedo Delgado Rafael Norman

01 de Junio de 2016

ESCOM

Instituto Politécnico Nacional
Escuela Superior de Computo

Procesador de texto amigable con el usuario

No. De registro: 16-2-006 **Junio del 2016**

Palabras clave:

Procesamiento de textos, Markdown, editor de textos, compiladores.

Resumen:

La integración continua de características avanzadas y especializadas en los procesadores de texto modernos ha tenido efectos negativos en la usabilidad de los mismos, investigadores, profesionistas y personas en general cuentan con la necesidad de redactar informes, reportes u otro tipo de documentos, en todo tipo de entornos laborales y académicos.

El presente documento contiene el análisis, diseño y desarrollo de un procesador de textos, que proporcione al usuario una manera sencilla para la creación y edición principalmente de reportes técnicos, se utilizará el lenguaje de marcado *Markdown* con el cual se busca conseguir la máxima facilidad en la edición y producción del documento.

Presenta:

Montalvo Lezama Mario[1]

Director:

Saucedo Delgado Rafael Norman

1. mario.mml.90@gmail.com

ESCUELA SUPERIOR DE CÓMPUTO
SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE FORMACIÓN INTEGRAL E INSTITUCIONAL

COMISIÓN ACADÉMICA DE TRABAJO TERMINAL

Ciudad de México a 08 de Junio de 2016.

DR. FLAVIO ARTURO SÁNCHEZ GARFIAS
PRESIDENTE DE LA COMISIÓN ACADÉMICA
DE TRABAJO TERMINAL
P R E S E N T E

Por medio del presente, se informa que el alumno del TRABAJO TERMINAL: 16-2-0006, titulado "Procesador de textos amigable con el usuario" concluyó satisfactoriamente su trabajo.

Los discos (DVDs) fueron revisados ampliamente por su servidor y corregido, cubriendo el alcance y el objetivo planteados en el protocolo original y de acuerdo a los requisitos establecidos por la Comisión que usted preside.

ATENTAMENTE

M. en C. Rafael Norman Saucedo Delgado
Director del Trabajo Terminal

Advertencia

“Este documento contiene información desarrollada por la Escuela Superior de Computo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convenga.”

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en:

La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

Índice General

Índice de figuras

1. Introducción

En el presente documento se describe el funcionamiento de un procesador de textos para la creación y edición de distintos tipos de documentos.

En el texto se detallan análisis, diseño y desarrollo de la aplicación, que proporcione al usuario una manera sencilla para la creación y edición principalmente de reportes técnicos, contiene los diagramas clases, secuencia y casos de uso que muestran el funcionamiento de cada uno de los módulos del procesador así como también los diagramas que describen la interacción con el usuario la herramienta.

1.1. Planteamiento del problema

En la actualidad las personas suelen tener la necesidad crear distintos tipos de documentos haciendo uso de herramientas que cuenten con características avanzadas y especializadas, la falta de procesadores de texto que conjunten facilidad de uso y calidad en el documento suele ser un problema para los usuarios a la hora de crear documentos.

1.2. Propuesta de solución

La propuesta de solución es realizar un procesador de textos, la herramienta está orientada a usuarios que cuenten con la necesidad de redactar reportes técnicos y requieran un procesador de textos sencillo y rápido de usar.

Se busca que el desarrollo de este procesador proporcione al usuario una manera sencilla para la creación y edición de textos ya que en la actualidad esta tarea suele ser tediosa o en algunos otros casos difícil.

Utilizando la sencillez que ofrece la sintaxis del lenguaje *Markdown* se busca conseguir la máxima claridad tanto en la forma de edición como en la producción, creando bajo el paradigma "lo que ves es lo que quieres decir" un procesador de textos que permita al usuario

concentrarse en el contenido y más que el formato del documento.

Haciendo uso de bibliotecas para la creación del procesador se busca proveer al usuario de funciones con las cuales pueda crear documentos, tales funciones como son: proveer distintos estilos, soporte básico de ecuaciones matemáticas, agregar bloques de código, enlaces, añadir imágenes entre otras.
El usuario podrá utilizar comandos sencillos para la creación de sus documentos, estos comandos serán interpretados por el sistema para proporcionar un formato al documento de salida.

1.3. Objetivo

Desarrollar un procesador de textos, para la creación, edición y modificación de reportes técnicos, ofreciendo al usuario una manera sencilla para la redacción de documentos, la cual permita conseguir la máxima claridad en la forma de edición y producción.

1.4. Estado del arte

A continuación se muestran publicaciones, aplicaciones comerciales y trabajos terminales relacionados y con características similares a la propuesta de solución descrita en este documento.

1.4.1. Trabajos terminales (TT):

De los trabajos terminales realizados en la Escuela Superior de Cómputo se encontraron los siguientes los cuales tienen alguna relación con la propuesta descrita en este documento:

- TT 2000-0268, Locución de textos con animación [1]: Software que presenta una imagen animada hablando un texto proporcionado por el usuario dando la entonación adecuada según el tipo de oración que este leyendo.

- TT 2012-B045, Aplicación Móvil con Realidad Aumentada de Apoyo a Turistas en Traducción de Textos [2]: El presente trabajo propone el diseño de una aplicación móvil para el auxilio de turistas en la tarea de traducción y comprensión de indicaciones textuales encontradas en establecimientos tales como museos u hoteles, mediante el uso de la tecnología de realidad aumentada.

En el TT 2000- 0268 tiene semejanza ya que se trabaja con texto para después utilizarlo para producir correctas entonaciones y deducir del significado de las palabras que lo forman. En el TT 2012-B045 mediante la lectura de indicaciones textuales se realiza un procesamiento de texto para indicarlo mediante la realidad aumentada.

1.4.2. Artículos publicados por la I.E.E.E:

Artículos publicados en la IEEE relacionados al Trabajo Terminal disponibles son:

- Plantilla de preparación de artículos técnicos en procesador de texto Word (Microsoft) para el Concurso Regional de Papers Estudiantiles [3]: Diseño de edición de un artículo técnico para el Concurso Regional de Papers Estudiantiles.
- LyX [4]: Es un procesador de documentos que fomenta la escritura con un enfoque basado en la estructura del documento (WYSIWYM) y no simplemente en su aspecto (WYSIWYG).

En el primer artículo mostrado provee una plantilla en formato docx la cual facilita la creación de artículos, esta plantilla se encuentra escrita en Word por lo tanto suele ser difícil mantener formato. Por otro lado LyX es un procesador el cual utiliza el lenguaje latex para la edición de documentos y en ocasiones suele ser complicado el uso de este lenguaje

1.4.3. Aplicaciones relacionadas

En la tabla 1 se pueden observar los principales procesadores de texto comerciales que existen en la actualidad.

Procesador de texto	Características	Ventajas	Desventajas
Microsoft® Word [5]	Convertido casi en un estándar de facto de referencia dado el elevado porcentaje de usuarios que lo utilizan.	Fácil de utilizar. Compartir contenido. Gran cantidad de herramientas fotográficas	Difícil mantener formato. Limitado para editar y manejar imágenes

WordPad [6]	Programa sencillo de procesamiento de texto el cual sirve para el tratamiento de textos.	Reconocimiento de voz. Almacenamiento y localización. Copiar bloques de texto. Búsqueda de palabras.	No tiene manejo de tablas. No tiene corrección de errores ortográficos. Soporta formato docx. No proporciona herramientas sofisticadas.
Libre Office Writer [7]	Programa para la elaboración de documentos complejos y por supuesto es compatible con Microsoft Word.	Es libre Compatible con MS Office (todas las versiones) Exportar los documentos a PDF	Difícil mantener formato. Ocupa demasiada RAM. No tiene 100% de compatibilidad de tablas.
Abiword [8]	La similitud de su interfaz con la del Microsoft Word. Además cuenta con muchas de las mismas prestaciones del Word.	Múltiples plugins Programas adjuntos con prestaciones.	Herramientas simples. Diseño pobre.
Texmaker [9]	Sistema de composición de textos, orientado a la creación de documentos escritos.	Alta calidad tipográfica. Concentrarse exclusivamente en el contenido. TeX, el motor de formateo de LaTeX, es totalmente libre y multiplataforma.	Procesar archivo. Sintaxis difícil de aprender. Es necesario tener instalado el compilador.
MarkPad [10]	Edición lado a lado con previsualización.	Facilita la edición de archivos Markdon. El usuario se concentra en el contenido.	Se limita a la sintaxis de Markdown. No permite ecuaciones. Diseño pobre.

Tabla 1. Aplicaciones comerciales relacionadas

En la tabla 1 se muestran que la mayoría de los procesadores comerciales usan el paradigma WYSIWYG para la creación de los documentos, por otro lado los Texmaker y MarkPad usan WYSIWYM esto facilita al usuario estructurar el documento dejando los aspectos visuales a cargo del sistema de exportación

Texmaker usa látex como lenguaje para la edición y es usado principalmente para la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones

matemáticas y MarkPad permite la edición de archivos *Markdown* limitándose a la sintaxis de este.

2. Marco teórico

Se presentan los aspectos teóricos relacionados con el trabajo terminal, que permiten ubicar en un contexto específico la comprensión de sus componentes.

2.1. Procesadores de texto

Un procesador de texto [11] es una aplicación informática que permite crear y editar documentos de texto en una computadora. Se trata de un software de múltiples funcionalidades para la redacción, con diferentes tipografías, tamaños de letra, colores, tipos de párrafos, efectos artísticos y otras opciones.

2.1.1. Funciones

Los procesadores de textos [11] brindan una amplia gama de funcionalidades, ya sean tipográficas, semánticas, organizativas o estéticas; con algunas variantes según el programa informático de que se disponga.

Como regla general, todos pueden trabajar con distintos tipos y tamaños de letras o caracteres, formato de párrafo y efectos artísticos; además de brindar la posibilidad de insertar tablas, imágenes u otros objetos gráficos dentro del texto. Como ocurre con la mayoría de las herramientas informáticas, los trabajos realizados en un procesador de textos pueden ser guardados en forma de archivos, usualmente llamados documentos, así como impresos a través de diferentes medios.

2.1.2. Paradigmas

Dentro de los procesadores de textos existen principalmente dos paradigmas los cuales son:

- WYSIWYG[11](del inglés What You See Is What You Get, que significa ‘lo que

ves es lo que obtienes'), en el que el aspecto final del documento es el que el usuario ve mientras lo edita.

- WYSIWYM [11] (del inglés What You See Is What You Mean, que significa 'lo que ves es lo que quieres decir'), donde el usuario se encarga de introducir los contenidos de forma estructurada siguiendo su valor semántico, en lugar de indicar su formato de representación final. La principal ventaja de este sistema es que se produce una total separación entre contenido y presentación.

2.2. Reportes técnicos

Un Reporte Técnico [12] es un informe de investigación científica o tecnológica que aborda un tema de investigación o desarrollo. Son válidos trabajos de investigación en general, recopilación de soluciones existentes o desarrollo de aplicaciones novedosas.

El reporte técnico de investigación es un documento que se utiliza para informar tanto los procedimientos como los resultados de una investigación en forma concisa y dentro de una estructura lógica, el objetivo del informe consiste en presentar la investigación y no la personalidad del autor; por eso el tono ha de ser impersonal y nunca se emplea la primera persona.

El documento debe contener:

- Portada
- Nombre de la institución y Escudo.
- Título (breve, de no más de 15 palabras, además debe reflejar el contenido del documento)
- Autor
- Adscripción del autor
- CA del autor (Si el trabajo está relacionado con algún CA)
- Fecha
- Nombre de quien recibe el reporte
- Clave del reporte (si ya fue aceptado)

2.3. Procesadores de lenguaje

Un compilador [13] es un programa que puede leer un programa en un lenguaje (el lenguaje fuente) y traducirlo en un programa equivalente en otro lenguaje (el lenguaje destino), como se muestra en la figura 1.

Una función importante del compilador es reportar cualquier error en el programa fuente que detecte durante el proceso de traducción.

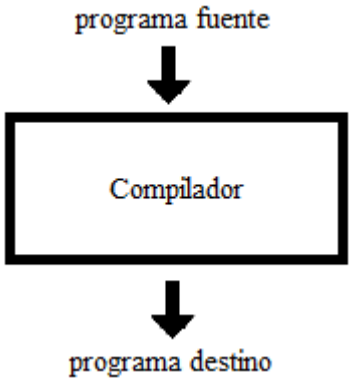


Fig. 1. Diagrama de compilador

Si el programa destino es un programa ejecutable en lenguaje máquina, entonces el usuario puede ejecutarlo para procesar las entradas y producir salidas (resultados); como se muestra en la figura 2.

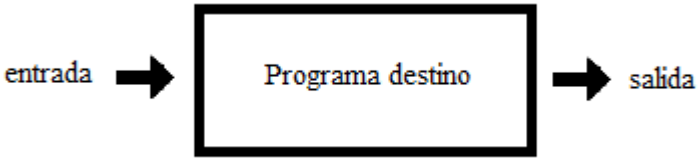


Fig. 2. Ejecución de programa de destino

Un intérprete es otro tipo común de procesador de lenguaje. En vez de producir un programa destino como una traducción, el intérprete nos da la apariencia de ejecutar directamente las operaciones especificadas en el programa de origen (fuente) con las entradas proporcionadas por el usuario, como se muestra en la figura 3.

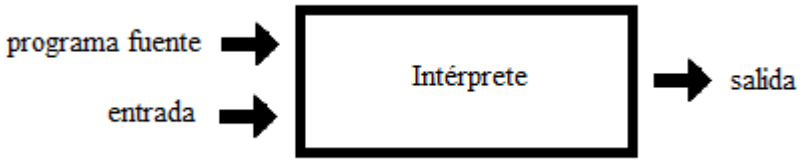


Fig. 3. Intérprete

El programa destino en lenguaje máquina que produce un compilador es, por lo general, más rápido que un intérprete al momento de asignar las entradas a las salidas. No obstante, por lo regular, el intérprete puede ofrecer mejores diagnósticos de error que un compilador, ya que ejecuta el programa fuente instrucción por instrucción.

2.3.1. La estructura de un compilador

La parte del análisis divide el programa fuente en componentes e impone una estructura gramatical sobre ellas. Después utiliza esta estructura para crear una representación intermedia del programa fuente. Si la parte del análisis detecta que el programa fuente está mal formado en cuanto a la sintaxis, o que no tiene una semántica consistente, entonces debe proporcionar mensajes informativos para que el usuario pueda corregirlo.

La parte del análisis también recolecta información sobre el programa fuente y la almacena en una estructura de datos llamada tabla de símbolos, la cual se pasa junto con la representación intermedia a la parte de la síntesis.

La parte de la síntesis construye el programa destino deseado a partir de la representación intermedia y de la información en la tabla de símbolos. A la parte del análisis se le llama comúnmente el front-end del compilador; la parte de la síntesis (propiamente la traducción) es el back-end. En la figura 4 se muestra una descomposición típica de un compilador en fases.

En la práctica varias fases pueden agruparse, y las representaciones intermedias entre las fases agrupadas no necesitan construirse de manera explícita. La tabla de símbolos, que almacena información sobre todo el programa fuente, se utiliza en todas las fases del compilador.

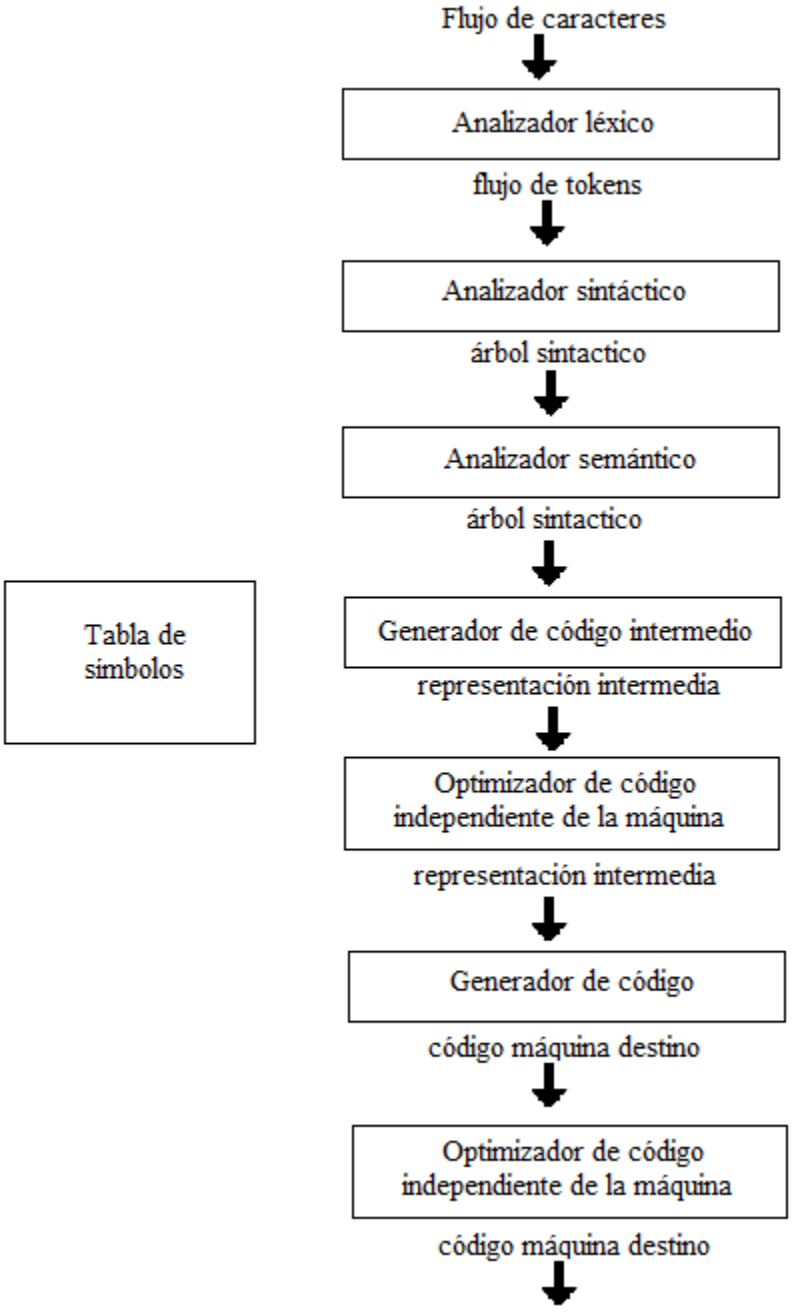


Fig. 4. Fases de un compilador

Algunos compiladores tienen una fase de optimización de código independiente de la máquina, entre el front-end y el back-end. El propósito de esta optimización es realizar transformaciones sobre la representación intermedia, para que el back-end pueda producir un mejor programa destino de lo que hubiera producido con una representación intermedia sin optimizar. Como la optimización es opcional, puede faltar una de las dos fases de optimización de la figura 4

2.3.2. Gramática

Las gramáticas [13] se utilizan para describir lenguajes. Existe una primera clasificación para las gramáticas, y esta estará en función a los tipos de lenguajes que genere, esto es ya sean lenguajes naturales o lenguajes formales.

Un **lenguaje natural** es como el Español o el Inglés o como cualquier otro lenguaje de comunicación entre personas, en donde la estructura de las frases, se describen por medio de una gramática que agrupa las palabras en categorías sintácticas tales como sujetos, predicados, frases preposicionales, etcétera. Estas construcciones gramaticales surgen como un intento de explicar las formas admitidas por el Lenguaje, con las cuales se construyen las frases, aunque en su definición se presentarán excepciones gramaticales.

Un **lenguaje Formal**, por el contrario surge a partir de su gramática, y por lo tanto no presenta excepciones en su definición. Esto es así, porque los Lenguajes Formales son los que se utilizan para que se comuniquen los hombres con las máquinas. De esta manera a partir de las gramáticas formales es como surgen los Lenguajes de Programación.

Es una forma de describir un lenguaje formal. La gramática permite generar cadenas a partir de un símbolo inicial y aplicando reglas que indican como ciertas combinaciones de símbolos pueden ser reemplazadas usando otras combinaciones de símbolos.

Una Gramática Formal G se compone de:

- Un conjunto finito de Símbolos No Terminales (N)
- Un conjunto finito de Símbolos Terminales (Σ)
- Un conjunto finito de Reglas de Producción (P)
    - Cada regla tiene la forma $\alpha \rightarrow \beta$, donde α y β pertenecen a $(\Sigma \cup N)^*$, la parte izquierda (α) debe contener al menos un símbolo no termina
- Un símbolo de Inicio (S), que pertenece a N

El lenguaje de una gramática G = (N, Σ, P, S), se denota como L(G), y se define como todas aquellas cadenas sobre Σ que pueden ser generadas inicando en el simbolo S, y aplicando las reglas de producción P, hasta reemplazar todos los símbolos no terminales.

2.4. Analizador Léxico

Un analizador léxico [13] es la primera fase de un compilador consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico.

La especificación de un lenguaje de programación a menudo incluye un conjunto de reglas que definen el léxico. Estas reglas consisten comúnmente en expresiones regulares que indican el conjunto de posibles secuencias de caracteres que definen un token o lexema. Otra función es relacionar los mensajes de error del compilador con el programa fuente.

En algunas ocasiones, los analizadores léxicos se dividen en una cascada de dos fases; la primera, llamada "examen", y la segunda, "análisis léxico". El examinador se encarga de realizar tares sencillas, mientras que el analizador léxico es el que realiza las operaciones complejas. En la figura 5 se puede observar funcionamiento general de un analizador léxico.

Fig. 5. Funcionamiento de analizador léxico

2.4.1. Aspecto del análisis léxico

Existen varias razones para dividir la fase de análisis de la compilación en análisis léxico y análisis sintáctico.

- Un diseño sencillo es quizá la consideración más importante. Separar el análisis léxico del análisis sintáctico a menudo permite simplificar una u otra de dichas fases.
- Se mejora la eficiencia del compilador. Un analizador léxico independiente permite construir un procesador especializado y potencialmente más eficiente para esta

función. Gran parte de tiempo se consume en leer el programa fuente y dividirlo en componentes léxicos. Con técnicas especializadas de manejo de buffer para la lectura de caracteres de entrada y procesamiento de componentes léxicos se puede mejorar significativamente el rendimiento de un compilador.

- Se mejora la transportabilidad del compilador. Las peculiaridades del alfabeto de entrada y otras anomalías propias de los dispositivos pueden limitarse al analizador léxico.

2.5. Analizador Sintáctico

Un analizador sintáctico [13] es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

El análisis sintáctico convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o árboles de sintaxis abstracta.

La forma más habitual de representar la sintaxis de un programa es el árbol de análisis sintáctico, y lo que hacen los analizadores sintácticos es construir una derivación por la izquierda o por la derecha del programa fuente, que en realidad son dos recorridos determinados del árbol de análisis sintáctico. En la figura 6 se puede observar funcionamiento general de un analizador sintáctico.

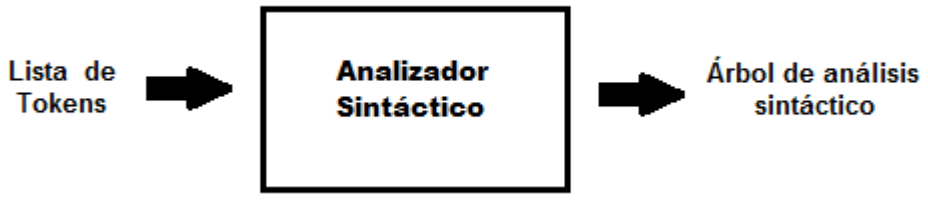


Fig. 6. Funcionamiento de analizador sintáctico.

2.5.1. Analizadores sintácticos LR

Los analizadores sintácticos LR [13], también conocidos como parser LR, son un tipo de analizadores para algunas gramáticas libres de contexto. Pertenece a la familia de los analizadores ascendentes, ya que construyen el árbol sintáctico de las hojas hacia la raíz. Utilizan la técnica de análisis por desplazamiento reducción.

Un analizador LR consta de:

- Un programa conductor
- Una entrada
- Una salida
- Una tabla de análisis sintáctico, compuesta de dos partes (ACCIÓN Y GOTO).

2.6. Herramientas de desarrollo

A continuación se definen algunos conceptos relacionados en el análisis e implementación del sistema.

2.6.1. Lenguajes de programación

Se presentan los lenguajes de programación para el desarrollo del sistema.

2.6.1.1. Python

Python [15] es un lenguaje de programación multiparadigma, que soporta programación orientada a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado y es dinámicamente tipado.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Fue creado a finales de los ochenta por Guido van Rossum en el Centro para las

Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica), en los Países Bajos, es capaz de manejar excepciones e interactuar con el sistema operativo Amoeba.

Una característica de Python es la resolución dinámica de nombres, que enlaza un método y un nombre de variable durante la ejecución del programa.

2.6.2. PLY

PLY [16] es una herramienta desarrollada por David Beazly, un desarrollador de software independiente de la ciudad de Chicago. Se trata básicamente de una implementación de Lex y Yacc originalmente en lenguaje C. Fue escrito por David Beazley.

A diferencia de Lex y Yacc en C que utiliza la técnica de análisis sintáctico LALR, PLY utiliza análisis sintáctico LR que puede incorporar grandes gramáticas fácilmente. PLY también tiene amplias instalaciones de depuración y reporte de errores.

2.6.3. HTML

HTML[17] significa "Lenguaje de Marcado de Hypertexto" por sus siglas en inglés "HyperText Markup Language", es un lenguaje que pertenece a la familia de los "lenguajes de marcado" y es utilizado para la elaboración de páginas web. El estándar HTML lo define la W3C (World Wide Web Consortium) y actualmente HTML se encuentra en su versión HTML5.

HTML no es un lenguaje de programación ya que no cuenta con funciones aritméticas, variables o estructuras de control propias de los lenguajes de programación, por lo que HTML genera únicamente páginas web estáticas, sin embargo, HTML se puede usar en conjunto con diversos lenguajes de programación para la creación de páginas web dinámicas.

Básicamente el lenguaje HTML sirve para describir la estructura básica de una página y organizar la forma en que se mostrará su contenido, además de que HTML permite incluir enlaces (links) hacia otras páginas o documentos.

HTML es un lenguaje de marcado descriptivo que se escribe en forma de etiquetas para definir la estructura de una página web y su contenido como texto, imágenes, entre otros, de modo que HTML es el encargado de describir (hasta cierto punto) la apariencia que

tendrá la página web.

Las etiquetas HTML son fragmentos de texto rodeados por corchetes angulares < >, que se utilizan para escribir código HTML, en HTML existen etiquetas de apertura y etiquetas de cierre, tienen la forma:

<etiqueta> </etiqueta>

Donde <etiqueta> es la etiqueta de apertura y </etiqueta> es la etiqueta de cierre indicada por la diagonal. HTML tiene definidas gran variedad de etiquetas para distintos usos.

Para desarrollar una página web en HTML es necesario crear un documento HTML. Básicamente un documento HTML es un archivo de texto que tienen la extensión .html o .htm, en este documento se escriben todo el texto y las etiquetas HTML necesarias para la creación de una página, al texto escrito en el documento HTML se le llama código HTML

2.6.4. CSS

CSS[18] es un lenguaje de estilo que define la presentación de los documentos HTML. CSS abarca cuestiones relativas a fuentes, colores, márgenes, líneas, altura, anchura, imágenes de fondo, posicionamiento avanzado y muchos otros temas.

Es posible usar HTML, para añadir formato a los sitios web; sin embargo, CSS ofrece más opciones y es más preciso y sofisticado. CSS está soportado por todos los navegadores hoy día.

2.6.5. Mathjax

MathJax [19] es una biblioteca javascript que permite visualizar fórmulas matemáticas en navegadores web, utilizando los lenguajes de marcado LaTeX o MathML. MathJax tiene licencia libre y soporta múltiples navegadores.

MathJax, que es descargado entre el contenido de una página web, analiza el contenido de esta página para buscar expresiones matemáticas (en lenguaje LaTeX o MathML) y las dibuja. Por lo tanto, MathJax no requiere la instalación de ningún tipo de software o de tipos de letras adicionales en el sistema del lector.

MathJax puede mostrar matemáticas utilizando una combinación de HTML y CSS o bien

puede utilizar MathML, si es que está soportado por el navegador. El método exacto que MathJax utiliza para componer las expresiones matemáticas está determinado por las características del navegador del usuario, por los tipos de letras disponibles en el sistema del usuario y por la configuración que se haya establecido.

En caso de usar HTML y maquetación CSS, MathJax maximiza la calidad de las matemáticas mediante el uso de tipos de letras matemáticas, si están disponibles. En caso de que no lo estén, se recurrie al uso de imágenes. En concreto, para los navegadores web más recientes, que pueden utilizar tipos de letras web, de hecho MathJax proporciona un amplio conjunto de tipos de letras web, que se descargan automáticamente, según sea necesario. Si el navegador no es compatible con tipos de letras web, MathJax comprueba si hay disponibles tipos de letras válidas en el sistema del usuario. MathJax puede usar notación matemática escrita en los lenguajes de marcado LaTeX o MathML. Debido a que MathJax está diseñado solamente para mostrar matemáticas y LaTeX es un lenguaje de diseño de documentos, MathJax sólo es compatible con el subconjunto de LaTeX que se usa para describir la notación matemática.

2.6.6. Markdown

Markdown [20] es un lenguaje de marcado ligero creado por John Gruber que trata de conseguir la máxima legibilidad y facilidad de publicación tanto en su forma de entrada como de salida, inspirándose en muchas convenciones existentes para marcar mensajes de correo electrónico usando texto plano.

En seguida se enlistan los comandos de formateo que permite Markdown como lenguaje de marcado:

Encabezados:

Los encabezados se generan cuando se encuentra signo numeral antes de texto

Encabezado h1
Encabezado h2
Encabezado h3
Encabezado h4
Encabezado h5
Encabezado h6

Citas:

Para citar solo es necesario escribir un signo “mayor que” antes del texto.

> La vida es muy corta para aprender Alemán. -Tad Marburg

Cursiva:

Agregar un asterisco al principio y al final de la o las palabras

énfasis

Negrita:

Agregar dos signos asteriscos al principio y al final de la o las palabras

énfasis fuerte

Código:

Se utiliza el acento grave para identificar código

`Código`

Listas:

- Numeradas
  1. Elemento en una lista enumerada u ordenada.
  2. Otro elemento
- No numeerdas
  * Elemento en una lista enumerada u ordenada
  * Elemento en una lista enumerada u ordenada

Enlaces:

[Texto del enlace aquí](URL "Título del enlace")

Imágenes:

![Texto alternativo](URL "Título de la imagen")

3. Metodología

Se utilizará la metodología basada en espiral de Boehm [21], dado que las actividades se acoplan a la forma de trabajo y planeación del sistema.

Estas actividades no tienen prioridad fija, se seleccionan en función del análisis de riesgo, comenzando por la iteración interior. Se tendrá además dos iteraciones de desarrollo donde se llevarán a cabo el análisis, diseño, construcción y pruebas de los prototipos 1 y 2.

- El prototipo 1 consistirá en el desarrollo del procesador de textos.
- El prototipo 2 consistirá en el desarrollo de la interfaz gráfica de usuario.

Al final de cada iteración se tendrá una etapa de análisis de riesgo para ajustar tiempo en caso de ser necesario.

4. Análisis de Sistema

En este capítulo se muestran los requerimientos del sistema y los diagramas que lo modelan. Así como la documentación de cada uno de ellos, los cuales describen la interacción del usuario con el sistema.

4.1. Requerimientos del Sistema

A continuación se enlistan los requerimientos funcionales y no funcionales del sistema además de los requerimientos que necesita el dispositivo para poder ejecutar la aplicación.

4.1.1. Requerimientos Funcionales

- El sistema debe permitir insertar texto.
- El sistema debe permitir insertar imágenes.
- El sistema debe permitir insertar ecuaciones.
- El sistema debe permitir cambiar estilos.
- El sistema debe permitir guardar el archivo fuente.
- El sistema debe permitir exportar a PDF el documento.
- El sistema debe permitir visualizar el documento final.

4.1.2. Requerimientos No funcionales

- La interfaz debe ser fácil de utilizar.
- La recarga de la pre visualización debe ser menos a 10 segundos.

4.2. Casos de Uso

En la figura 7 se muestra el diagrama de casos de uso de la interacción entre el usuario y el procesador de textos.

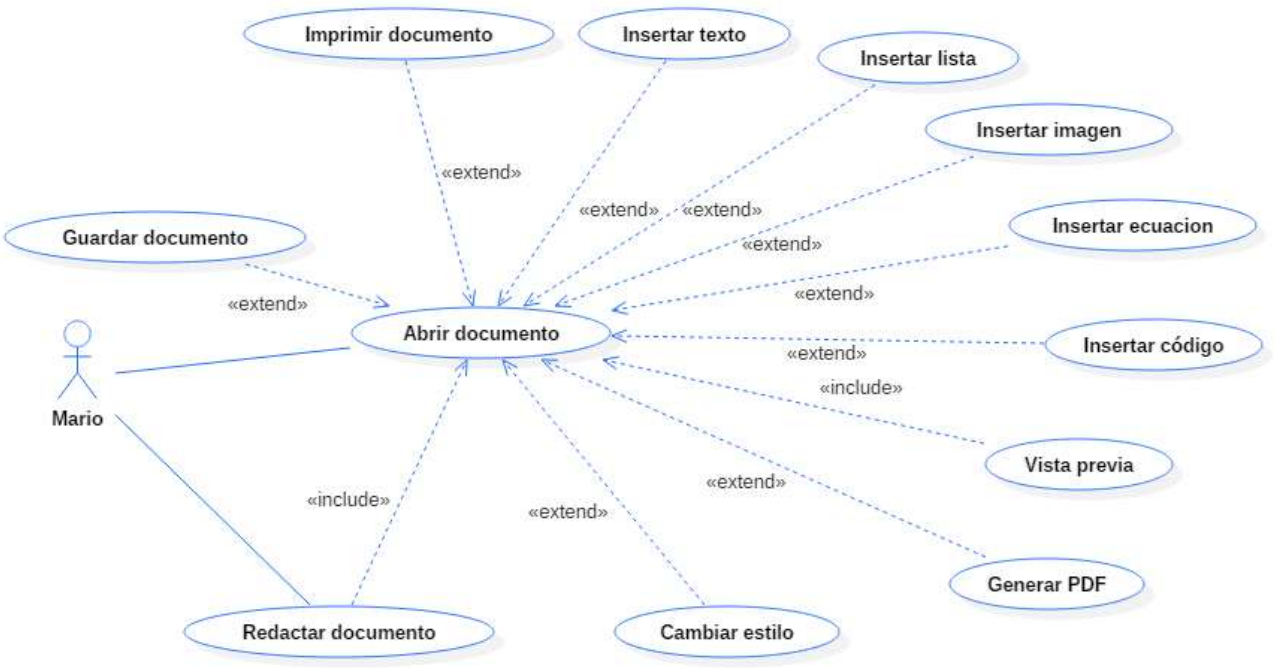


Fig. 7. Caso de uso de la interacción entre el usuario y el procesador de texto.

4.2.1. CU-01: Abrir Documento

Caso de uso	Abrir documento
Actor	Usuario
Propósito	Hacer uso de un documento.
Resumen	El usuario selecciona el documento que quiere usar.
Entradas	Nombre del documento
Salidas	Documento editado
Pre-condiciones	El documento sea del tipo .md
Errores	Error al abrir el documento

Trayectoria principal

1. El usuario selecciona el documento
2. El sistema verifica que el documento sea .md
3. El sistema abre el documento [T. Alt. A]
4. El sistema muestra el contenido del documento
5. Fin

Trayectoria alternativa A

Condición: Exista un error al abrir el documento

1. El sistema muestra una alerta de error
2. Fin

4.2.2. CU-02: Redactar Documento

Caso de uso	Redactar documento
Actor	Usuario
Propósito	Crear un documento.
Resumen	El usuario crea un documento nuevo
Salidas	Nuevo documento

Trayectoria principal

1. El usuario selecciona “redactar documento”
2. El sistema crear el documento
3. Ir al CU-01
4. Fin

4.2.3. CU-03: Guardar Documento

Caso de uso	Guardar documento
Actor	Usuario
Propósito	Guardar un documento abierto.
Resumen	El usuario guarda un documento.
Entradas	Ruta del documento, Nombre del documento
Salidas	Estado del documento
Pre-condiciones	No exista un documento con el mismo nombre
Errores	Error al guardar el documento

Trayectoria principal

1. El usuario selecciona “Guardar documento”
2. El sistema muestra la pantalla de Guardar documento
3. El usuario selecciona la ruta donde se guardará el documento
4. El usuario ingresa el nombre del documento
5. El sistema guarda el documento [T. Alt. A]
6. Fin

Trayectoria alternativa A

Condición: Exista un error al guardar el documento.

1. El sistema muestra una alerta de error
2. Fin

4.2.4. CU-04: Imprimir documento

Caso de uso	Imprimir documento
Actor	Usuario
Propósito	Imprimir un documento
Resumen	El usuario imprime un documento.
Entradas	Ruta del documento, Nombre del documento
Salidas	Estado del documento
Errores	Error al conectar con la impresora, Error al imprimir el documento

Trayectoria principal

1. El usuario selecciona “Imprimir documento”
2. El sistema muestra la pantalla de imprimir documento
3. El sistema se conecta con la impresora [T. Alt. A]
4. El sistema imprime el documento [T. Alt. B]

Trayectoria alternativa A

Condición: Exista un error al conectar con la impresora

1. El sistema muestra una alerta de error
2. Fin

Trayectoria alternativa B

Condición: Exista un error al imprimir el documento.

1. El sistema muestra una alerta de error
2. Fin

4.2.5. CU-05: Generar PDF

Caso de uso	Generar PDF
Actor	Usuario
Propósito	Generar PDF del documento
Resumen	El usuario genera el PDF del documento
Entradas	Ruta del PDF, Nombre del PDF
Salidas	Estado del PDF
Pre-condiciones	No esté vacío el documento
Errores	Error al generar el PDF Error al guardar el PDF

Trayectoria principal

1. El usuario selecciona “Generar PDF”
2. El sistema comprueba que el documento no esté vacío
3. El sistema muestra la pantalla Generar PDF
4. El usuario ingresa la ruta donde se guardará el PDF
5. El usuario ingresa el nombre del PDF
6. El sistema genera el PDF [T. Alt. A]
7. El sistema guarda el PDF en la ruta especificada [T. Alt. B]
8. Fin

Trayectoria alternativa A

Condición: Exista un error al generar el PDF

1. El sistema muestra una alerta de error
2. Fin

Trayectoria alternativa B

Condición: Exista un error al guardar el PDF

1. El sistema muestra una alerta de error
2. Fin

4.2.6. CU-06: Ver Vista Previa

Caso de uso	Ver vista previa
Actor	Sistema
Propósito	Generar una vista previa del PDF
Resumen	El sistema genera una vista previa del PDF cada que el usuario lo modifica
Entradas	El archivo .md
Salidas	Vista previa
Errores	Error al generar la vista previa

Trayectoria principal

1. El sistema espera a que el usuario modifique el documento
2. El usuario modifica el documento
3. El sistema genera la vista previa del PDF [T. Alt. A]
4. El sistema muestra la vista previa
5. Fin

Trayectoria alternativa A

Condición: Exista un error al generar la vista previa.

1. El sistema muestra una alerta de error
2. Fin

4.2.7. CU-07: Insertar texto

Caso de uso	Insertar texto
Actor	Usuario
Propósito	Insertar texto en el documento
Resumen	El usuario inserta texto en el documento
Entradas	El documento
Salidas	El documento modificado

Trayectoria principal

1. El usuario ingresa el texto
2. El sistema modifica el documento
3. Ir al CU-06 en el paso 3
4. Fin

4.2.8. CU-08: Insertar ecuación

Caso de uso	Insertar ecuación
Actor	Usuario
Propósito	Insertar ecuación en el documento
Resumen	El usuario inserta ecuación en el documento
Entradas	El documento
Salidas	El documento modificado

Trayectoria principal

1. El usuario ingresa el ecuación
2. El sistema modifica el documento
3. Ir al CU-06 en el paso 3
4. Fin

4.2.9. CU-09: Insertar código

Caso de uso	Insertar código
Actor	Usuario
Propósito	Insertar código en el documento
Resumen	El usuario inserta código en el documento
Entradas	El documento
Salidas	El documento modificado

Trayectoria principal

1. El usuario ingresa el código
2. El sistema modifica el documento
3. Ir al CU-06 en el paso 3
4. Fin

4.2.10. CU-10: Insertar lista

Caso de uso	Insertar lista
Actor	Usuario
Propósito	Insertar una lista en el documento
Resumen	El usuario inserta una lista en el documento
Entradas	El documento
Salidas	El documento modificado

Trayectoria principal

1. El usuario ingresa una lista
2. El sistema modifica el documento insertando la lista
3. Ir al CU-06 en el paso 3
4. Fin

4.2.11. CU-11: Insertar Imagen

Caso de uso	Insertar imagen
Actor	Usuario
Propósito	Insertar una imagen en el documento
Resumen	El usuario inserta una imagen en el documento
Entradas	El documento
Salidas	El documento modificado

Trayectoria principal

1. El usuario selecciona “Insertar imagen”
2. El sistema muestra la pantalla insertar imagen
3. El usuario selecciona la imagen que desea insertar
4. El sistema modifica el documento insertando la imagen
5. Ir al CU-06 en el paso 3
6. Fin

4.2.12. CU-12: Cambiar Estilo

Caso de uso	Cambiar estilo
Actor	Usuario
Propósito	Cambiar el estilo del documento
Resumen	El usuario cambia el estilo del documento
Entradas	El documento
Salidas	El documento modificado

Trayectoria principal

1. El usuario selecciona “Cambiar estilo”
2. El sistema muestra los estilos disponibles
3. El usuario escoge un estilo
4. El sistema cambia el estilo del documento
5. Ir al CU-06 en el paso 3
6. Fin

5. Diseño del sistema

En esta sección se muestran los diagramas y la documentación del diseño del sistema mostrando los diferentes diagramas de los componentes del mismo.

5.1. Arquitectura del Software

En la figura 8 se muestra el diagrama de la arquitectura del sistema, en donde se explica el proceso que se lleva a cabo en el procesador de textos.

El usuario introduce texto en el editor generando en este momento el archivo fuente, en seguida el texto es procesado por el compilador y haciendo uso de la librería Mathjax, se genera el código objeto que con ayuda de un componente se puede muestra la pre visualización del documento final.

En todo momento que usuario puede ver el documento que se está generando, además de poder exportarlo en formato PDF.

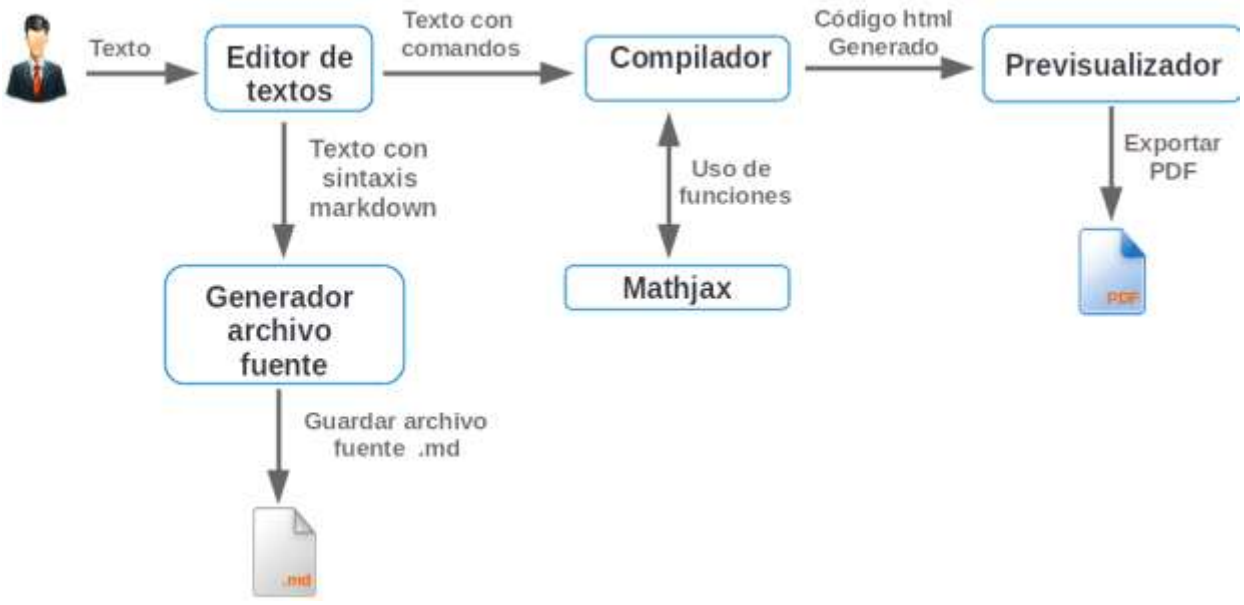


Fig. 8. Diagrama de la arquitectura de sistema

5.2. Diagramas de secuencia

En la figura 9 se muestra el diagrama de secuencia de vista previa. El usuario crea un documento donde escribe texto con la sintaxis que se especifica, posteriormente este se manda a compilador de markdown donde se analiza el texto de entrada y se genera el lenguaje de salida, generando así el procesador el documento de salida en el componente de pre visualización.

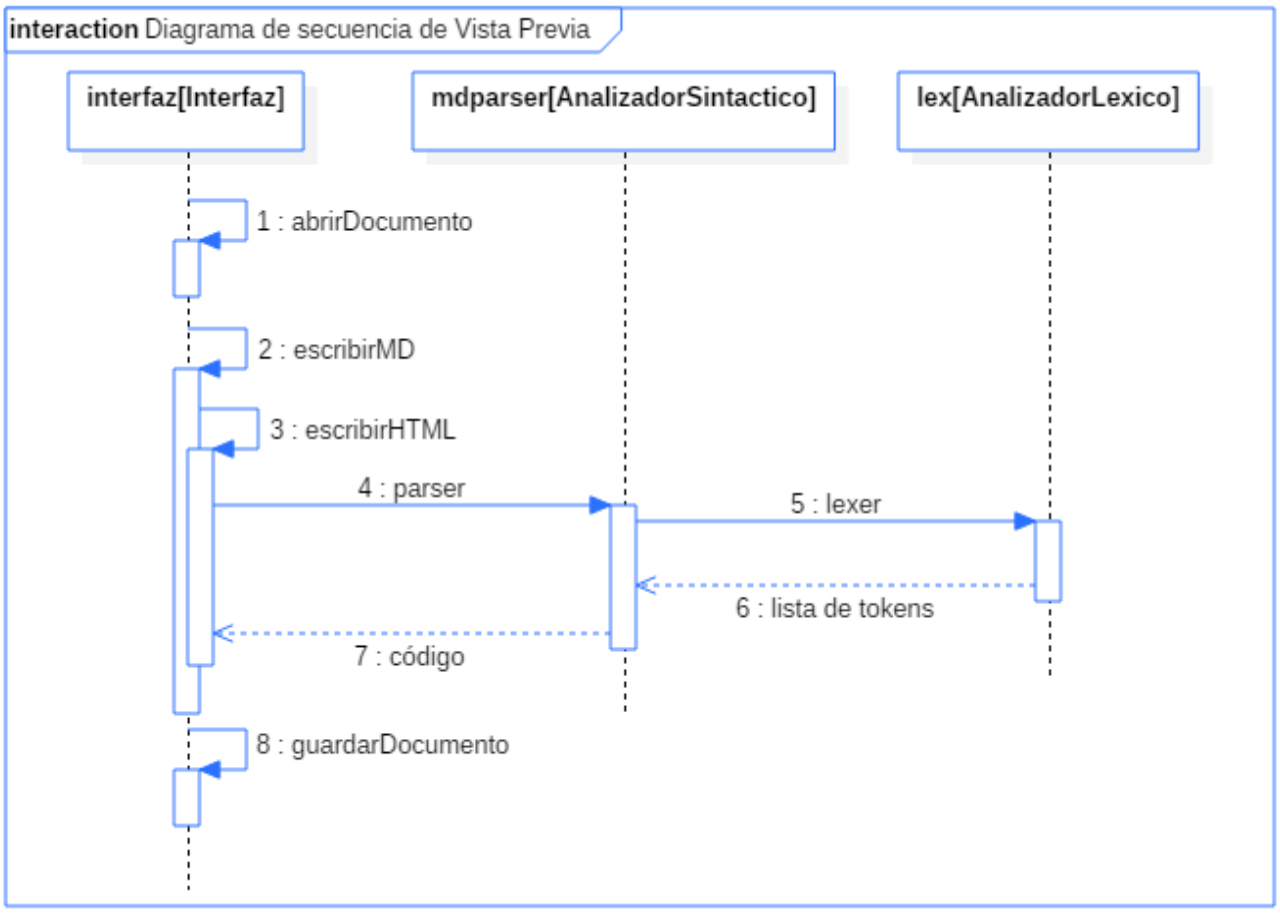


Fig. 9. Diagrama de secuencia vista previa.

5.3. Diagramas de clases

En la figura 10 se muestra el diagrama de clases del procesador de textos dónde se puede observar cómo está compuesta la aplicación. Se tiene como clase principal "Interfaz" en donde se escriben los tipos de archivos que se requieren para que el procesador funcione correctamente.

En la clase analizador léxico se tiene la definición de los TOKENS y de las expresiones regulares para el compilador del lenguaje., por último la clase analizador sintáctico en donde se definen cada una de las reglas gramaticales y se genera el código objeto.

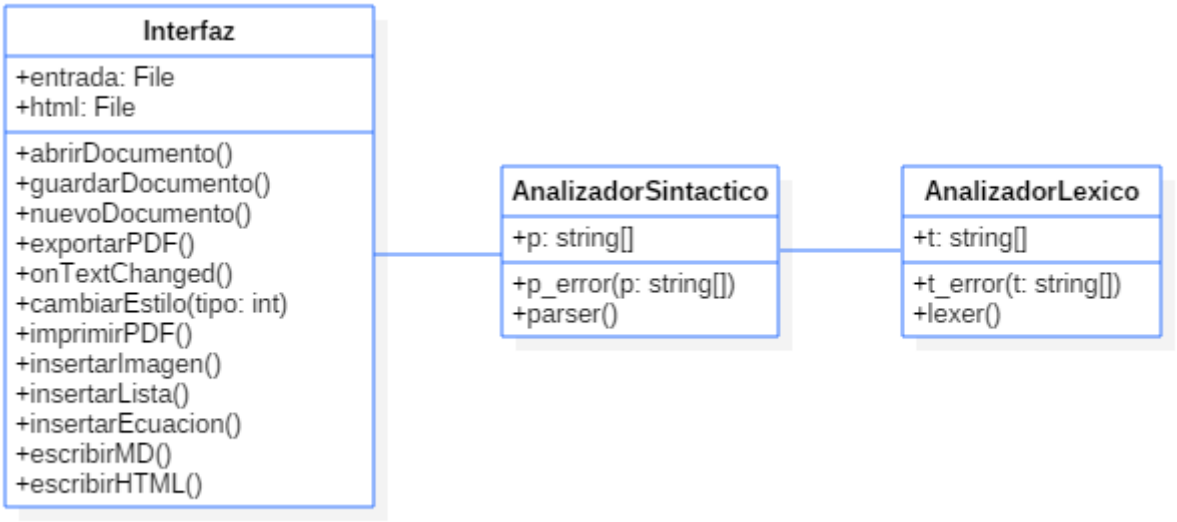


Fig. 10. Diagrama de Clases.

5.4. Interfaz

En la figura 11 se muestra el diseño de la interfaz de usuario de la aplicación, en donde se buscó principalmente las siguientes características:

En la parte izquierda muestra

- Facilidad de uso.
- Facilidad de aprendizaje.
- Facilidad de comprensión.
- Diseño ergonómico mediante el establecimiento de menús e iconos de fácil acceso.
- Las operaciones sean rápidas.

En la figura 11 se muestra del lado izquierdo el *editor de textos* en donde usuario puede ingresar el texto con los comandos de la sintaxis, del lado derecho se encuentra el *pre visualizador* en el cual el usuario podrá ver en todo momento el documento final y en la parte superior tenemos el *menú* de opciones para la creación y edición del documento.

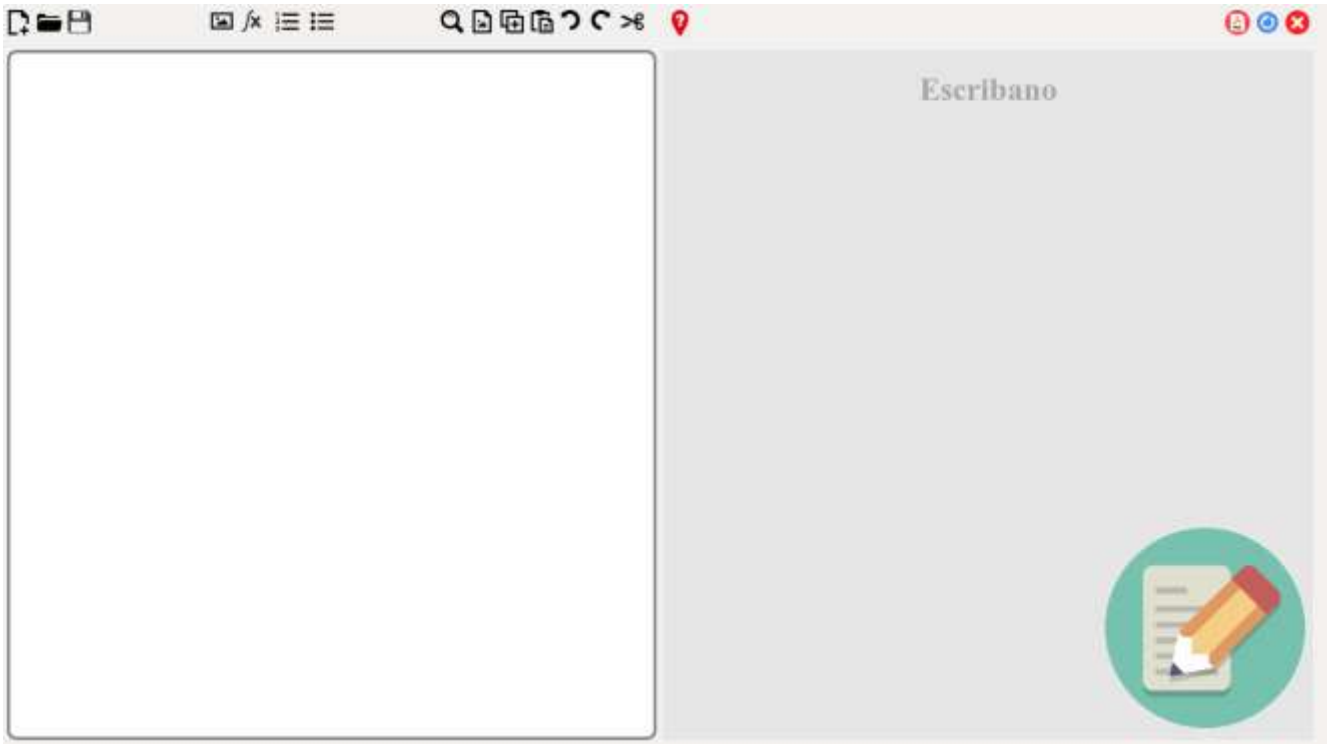


Fig. 11. Interfaz.

En la figura 12 se muestra el editor de texto, en donde se indican cada una de las funciones que este ofrece.

Funciones para el manipular el archivo:

- Abrir fuente archivo existente
- Crear un nuevo archivo
- Guardar el archivo fuente

Funciones para la edición del documento:

- Insertar ecuación
- Insertar imagen
- Insertar lista numerada
- Insertar lista no numerada

Funciones del editor

- Pegar
- Copiar
- Deshacer
- Cortar
- Hacer

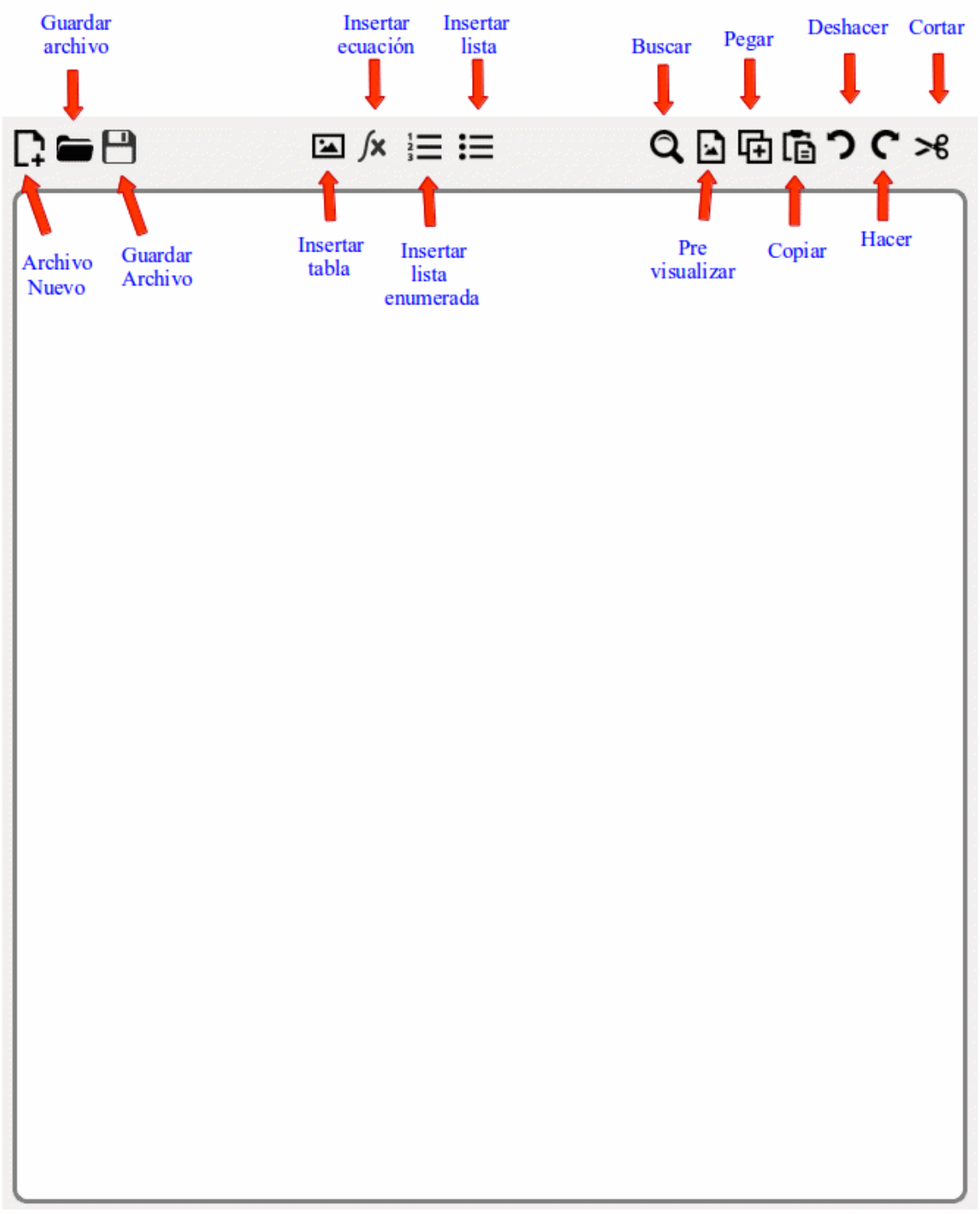


Fig. 12. Editor de textos.

En la figura 13 se muestra el componente para pre visualizar el documento final con formato, en donde se indican las funciones de las cuales el usuario puede hacer uso.

Una vez que se tiene vista previa es posible hacer uso de las siguientes funciones:

- Exportar el documento en formato PDF
- Recargar el pre visualizador.

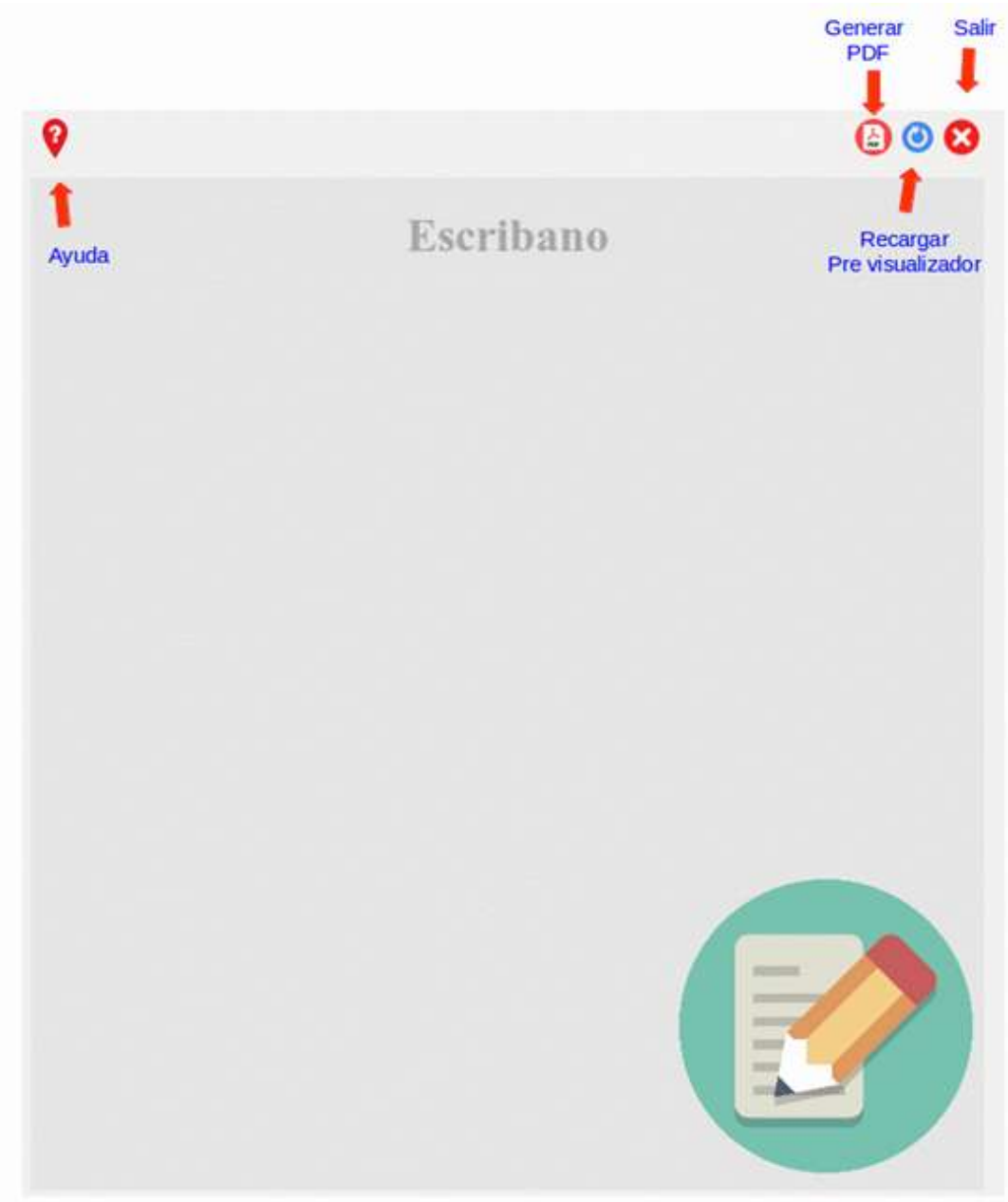


Fig. 13. Pre visualizador.

6. Desarrollo

A continuación se detalla desarrollo de cada una de las partes del sistema.

6.1. Desarrollo del compilador

El desarrollo del compilador se realizó con la biblioteca PLY la cual permite la implementación de un analizador léxico y un analizador sintáctico. PLY se implementa completamente en Python y utiliza el análisis sintáctico LR muy adecuado para gramáticas más grandes y proporciona una extensa comprobación de errores.

En la figura 14 se muestra las fases que fueron consideradas para el diseño del compilador,

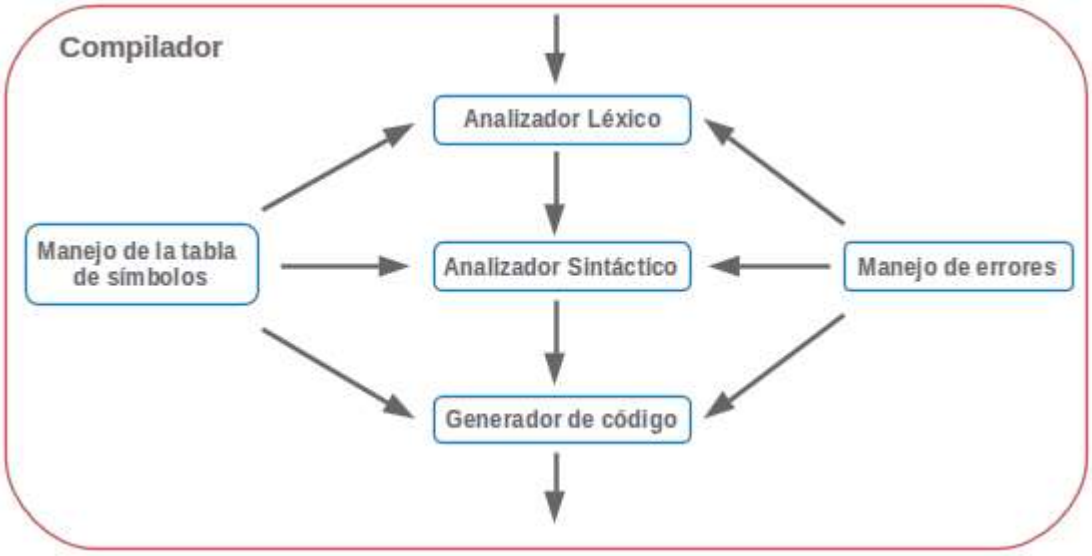


Fig. 14. Fases de compilador

A continuación se describe la implementación de las dos fases del compilador tales como son: analizador léxico y el analizador sintáctico.

6.1.1. Analizador léxico

Un analizador léxico es el componente encargado de identificar los tokens válidos en un código de entrada y emitir errores de ser necesario.

La función principal de un *lexer* es tomar un flujo de caracteres entrada y devolver un flujo de *tokens* como salida. Primero se genera una tupla con todos los nombres de los tokens que se desean reconocer.

El analizador léxico informa de cada *token* encontrado, en que línea y columna exactamente fue extraído el *token*, además del índice del *token*, el tipo y el lexema que lo representa.

En la figura 15 se muestra la salida y la entrada del analizador léxico.

Fig. 15. Funcionamiento del analizador léxico.

6.1.2. Analizador Sintáctico

La siguiente fase del compilador es el analizador sintáctico con el cual se busca dar orden a los tokens generados por el analizador léxico.

Esto lo conseguimos con una gramática, se utiliza el análisis sintáctico LALR, que es razonablemente eficiente y muy adecuado para gramáticas más grandes. Cuando se forman las reglas especificadas en el analizador se genera el código de salida las cuales con etiquetas de HTML para posteriormente crear el documento. En el capítulo 9 se muestran las principales reglas gramaticales que se han definido

En la figura 16 se muestra la salida y la entrada del analizador sintáctico.

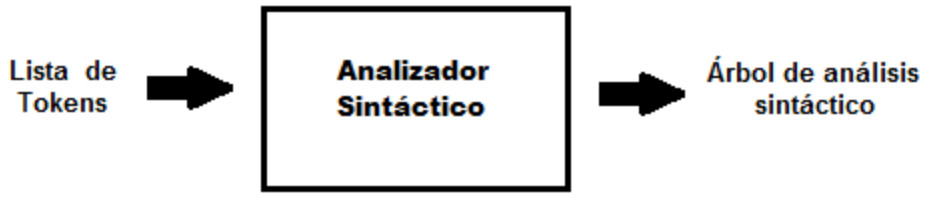


Fig. 16. Funcionamiento del analizador sintáctico.

6.1.3. Generación de código

En esta fase del compilador se genera el código objeto que se utiliza para la escritura del documento final, el cual son etiquetas HTML que posteriormente aplicando hojas de estilo CSS serán mostradas en el pre visualizador.

En la figura 17 se muestra la salida y la entrada del generador de código.

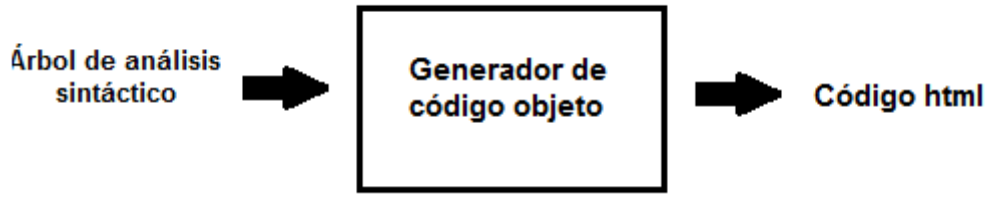


Fig. 17. Funcionamiento del generador de código.

6.2. Creación del documento

La escritura del documento final se realiza través del código generado por el compilador en la fase de generador de código; construyendo con el código HTML y con hojas de estilo CSS el formato final del documento en formato PDF.

Dentro del compilador se hace uso de la biblioteca Mathjax, la cual ayuda a dar formato a las ecuaciones que el usuario desea ingresar.

En la figura 18 se muestra este proceso que se lleva acabo para escribir el documento.

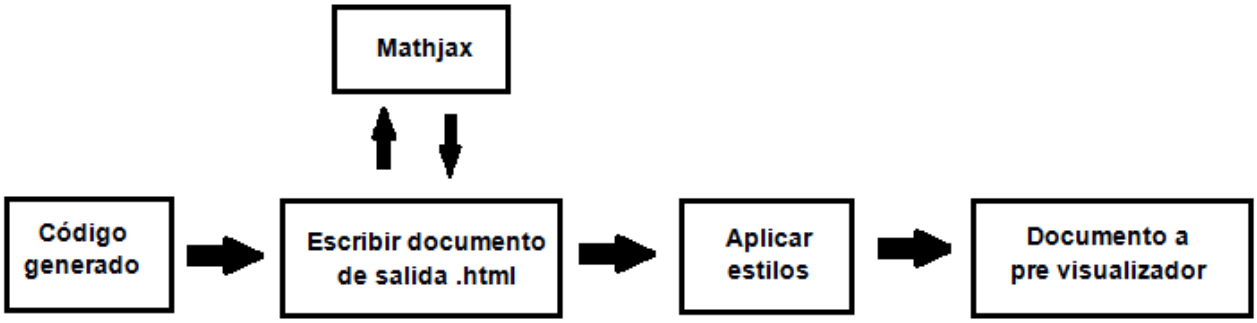


Fig. 18. Escribir documento

6.3. Sintaxis del procesador

A continuación se detalla la sintaxis que permite el procesador de textos para la creación y modificación de documentos:

6.3.1. Encabezados

Los encabezados se escriben con el signo **#** al principio y al final del texto que se quieren resaltar, el número de signos **#** indican el tamaño del encabezado.

Existen seis tamaños de encabezados en donde un signo # representa el más grande de estos y teniendo como limite la profundidad de seis encabezados.

Encabezado h1
Encabezado h2
Encabezado h3
Encabezado h4
Encabezado h5
Encabezado h6

En la figura 19 se muestra un ejemplo de los seis encabezados que soporta el procesador

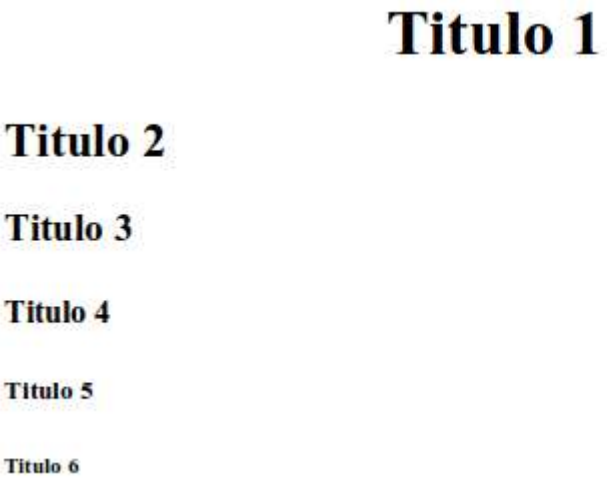


Fig. 19. Ejemplo de encabezados.

6.3.2. Citas

El uso de citas en el procesador de textos se hace con el operador:

> **Esto es una cita**

En la figura 20 se muestra el resultado de la cita en el pre visualizador.

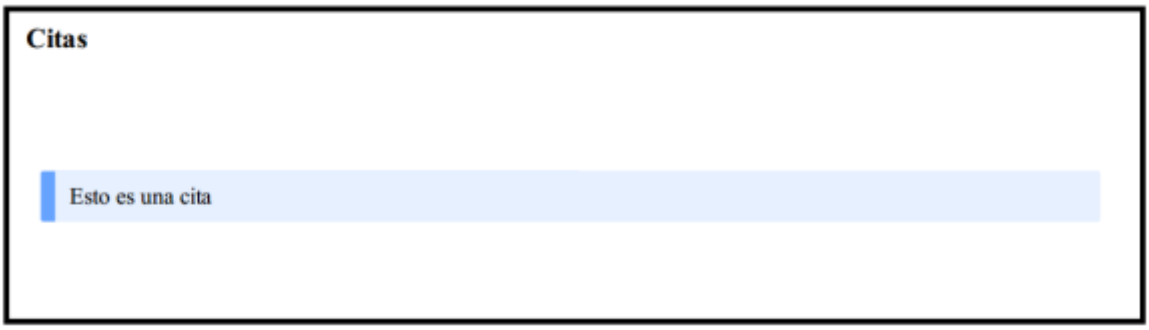


Fig. 20. Ejemplo de cita.

6.3.3. Cursiva:

Para negritas es necesario agregar un signo asterisco al principio y al final del texto que se requiera enfatizar.

*** texto que se requiere enfatizar ***

6.3.4. Negrita

Para negritas es necesario agregar dos signos asteriscos al principio y al final del texto que se requiera enfatizar.

**** texto que se requiere enfatizar ****

6.3.5. Subrayado

Para el texto subrayado es necesario agregar guion bajo al principio y al final del texto que se requiera enfatizar.

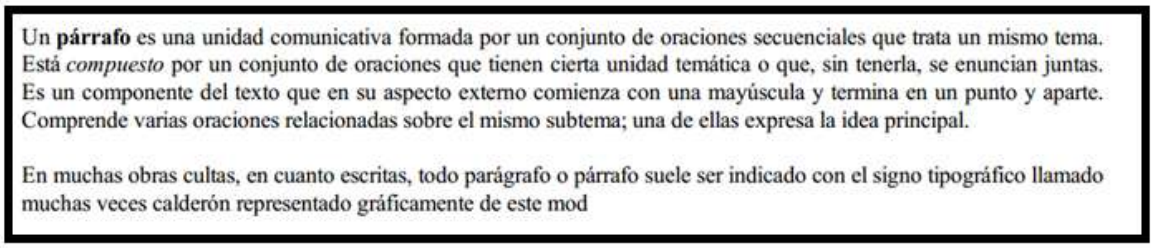
_ texto que se requiere subrayar_

6.3.6. Cursiva y negrita

Para negritas es necesario agregar dos signos asteriscos al principio y al final del texto que se requiera enfatizar.

***** texto que se requiere enfatizar *****

En la figura 21 se muestra el resultado de un párrafo en el pre visualizador aplicando varios énfasis dentro del texto.


Un **párrafo** es una unidad comunicativa formada por un conjunto de oraciones secuenciales que trata un mismo tema. Está *compuesto* por un conjunto de oraciones que tienen cierta unidad temática o que, sin tenerla, se enuncian juntas. Es un componente del texto que en su aspecto externo comienza con una mayúscula y termina en un punto y aparte. Comprende varias oraciones relacionadas sobre el mismo subtema; una de ellas expresa la idea principal.

En muchas obras cultas, en cuanto escritas, todo parágrafo o párrafo suele ser indicado con el signo tipográfico llamado muchas veces calderón representado gráficamente de este mod

Fig. 21. Ejemplo de párrafo.

6.3.7. Listas:

Existen dos tipos de listas:

- Las sintaxis de tipos de lista numeradas se define como:
  1. Elemento en una lista enumerada u ordenada.
  2. Otro elemento

- Las sintaxis de listas no numeradas se define como:
  * Elemento en una lista enumerada u ordenada
  * Elemento en una lista enumerada u ordenada

En la figura 22 se muestra el resultado de una lista en el pre visualizador.

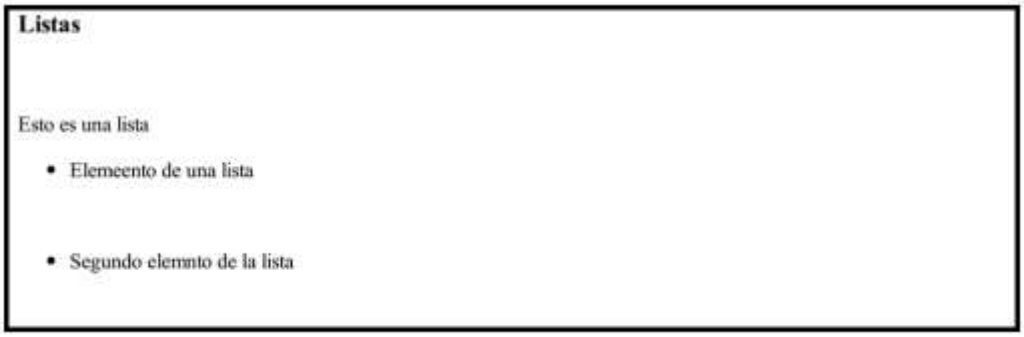


Fig. 22. Ejemplo de lista no enumerada.

6.3.8. Enlaces:

El uso de enlaces en el procesador de textos tiene la siguiente sintaxis:

[Esto es un enlace](URL)

En la figura 23 se muestra el resultado de un enlace en el pre visualizador.

Esto es una en lace

Fig. 23. Ejemplo de enlace

6.3.9. Imágenes:

El uso de imágenes en el procesador de textos tiene la siguiente sintaxis:

![Titulo de la imagen](URL)

Para especificar el tamaño de la imagen solo basta con poner las medidas seguidas de la url:

![Titulo de la imagen](URL)(200,200)

En la figura 24 se muestra el resultado de una insertar una imagen

Fig. 24. Ejemplo de imagen con título

6.3.10. Ecuaciones

Para una ecuación es necesario agregar dos signos "**$**" al principio y al final de cada uno de los operadores que se requieran utilizar.

$$ x = mx + b $$

En donde " **x = mx+ b"** será la ecuación mostrada como resultado. A continuación se muestran diferentes ejemplos de ecuaciones que soporta el procesador de textos

- **Ecuación general**
  En el editor de textos se escribe:

  $$ ax + by + c = 0 $$

  En la figura 25 se muestra la ecuación general de en el pre visualizador:

$$ax + by + c = 0$$

Fig. 25. Ejemplo de ecuación general

- **Ecuaciones**
  En el editor de textos se escribe:

$$ y = \underbrace{f(1)}_{parte\ 1} + \overbrace{f(2)}^{parte\ 2} $$

En la figura 26 se muestra la ecuación de en el pre visualizador:

$$y = \underbrace{f(1)}_{parte\ 1} + \overbrace{f(2)}^{parte\ 2}$$

Fig. 26. Ejemplo de ecuación

- **Símbolos**
  En el editor de textos se escribe:

$$ \alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \lambda $$

En la figura 27 se muestra los símbolos en el pre visualizador:

$$\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \lambda$$

Fig. 27. Ejemplo de signos

En el editor de textos se escribe:

$$ \mu, \nu, \xi, \pi, \rho, \sigma, \tau, \upsilon, \phi, \chi, \psi, \omega $$

En la figura 28 se muestra los símbolos en el pre visualizador:

$$\mu, \nu, \xi, \pi, \rho, \sigma, \tau, \upsilon, \phi, \chi, \psi, \omega$$

Fig. 28. Ejemplo de signos.

En el editor de textos se escribe:

$$ \Gamma, \Delta, \Theta, \Lambda, \Xi, \Pi, \Sigma, \Upsilon, \Phi, \Psi, \Omega $$

En la figura 29 se muestra los símbolos en el pre visualizador:

$$\Gamma, \Delta, \Theta, \Lambda, \Xi, \Pi, \Sigma, \Upsilon, \Phi, \Psi, \Omega$$

Fig. 29. Ejemplo de signos.

En el editor de textos se escribe:

$$ \varepsilon, \vartheta, \varpi, \varrho, \varsigma, \varphi $$

En la figura 30 se muestra los símbolos en el pre visualizador:

$$\varepsilon, \vartheta, \varpi, \varrho, \varsigma, \varphi$$

Fig. 30. Ejemplo de signos

En el editor de textos:

$$ \rightarrow \leftarrow \longrightarrow \longleftarrow $$

En la figura 27 se muestra las flechas en el pre visualizador:

Fig. 31. Ejemplo de flechas.

- **Derivadas**
  En el editor de textos se escribe:

  $$ y' = \dot{y} = \frac{\partial y}{\partial t} $$

  En la figura 32 se muestra un ejemplo de derivada de en el pre visualizador:

$$y' = \dot{y} = \frac{\partial y}{\partial t}$$

Fig. 32. Ejemplo de derivadas.

- **Fracciones**
  En el editor de textos se escribe:

  $$ \frac{x^2}{x^2 + y^2} + \frac15 + \frac1{x} $$

  En la figura 33 se muestra un ejemplo de fracciones de en el pre visualizador:

$$\frac{x^2}{x^2 + y^2} + \frac{1}{5} + \frac{1}{x}$$

Fig. 33. Ejemplo de fracciones.

- **Índices**
  En el editor de textos se escribe:

  $$ y(x_{i}) = 4 + x_{i}^{2} $$

En la figura 34 se muestra un ejemplo de índices de en el pre visualizador:

$$y(x_i) = 4 + x_i^2$$

Fig. 34. Ejemplo de índices.

- **Sumatoria**
  En el editor de textos se escribe:

  $$\sum_{i=0}^n i^2 =\frac{(n^2+n)(2n+1)}{6}$$

  En la figura 35 se muestra un ejemplo de sumatoria de en el pre visualizador:

$$\sum_{i=0}^{n} i^2 = \frac{(n^2+n)(2n+1)}{6}$$

Fig. 35. Ejemplo de sumatoria.

- **Integral**
  En el editor de textos se escribe:

  $$ \iiint\limits_A f(x,y,z)\;dx\;dy\;dz $$

  En la figura 36 se muestra un ejemplo de integral de en el pre visualizador:

$$\iiint\limits_A f(x,y,z)\;dx\;dy\;dz$$

Fig. 36. Ejemplo de integral.

- **Matriz**
  En el editor de textos se escribe:

$$ \begin{pmatrix} x & y \\ z & v \end{pmatrix}$$

En la figura 37 se muestra un ejemplo de matriz de en el pre visualizador:

$$\begin{pmatrix} x & y \\ z & v \end{pmatrix}$$

Fig. 37. Ejemplo de matriz.

- **Producto escalar**
  En el editor de textos se escribe:

  $$a \cdot b \times x$$

  En la figura 38 se muestra un ejemplo de producto escalar de en el pre visualizador:

$$a \cdot b \times x$$

Fig. 38. Ejemplo de producto escalar.

- **Transformada de Laplace**
  En el editor de textos se escribe:

  $$\mathscr{L}\{f(t)\} = F(s)$$

- En la figura 39 se muestra un ejemplo de Transformada de Laplace en el pre visualizador:

$$\mathscr{L}\{f(t)\} = F(s)$$

Fig. 39. Ejemplo de transformada de la Laplace.

- **Limite**
  En el editor de textos:

  $$ \lim_{x \to \infty} \frac{\sin(x)}{x} = 0 $$

  En la figura 40 se muestra un ejemplo de límites en el pre visualizador:

$$\lim_{x \to \infty} \frac{\sin(x)}{x} = 0$$

Fig. 40. Ejemplo de límite.

6.3.11. Resaltado de código

Si desea obtener resaltado de códigos de lenguajes de programación es necesario insertar tres signos ”dos puntos” (:::) el principio y al final de código.

A continuación se muestran ejemplos de resaltado de código en diferentes lenguajes de programación

- **Códigos en lenguaje Python**
  En el editor de textos se escribe:

  :::

```
import sys
import os
from PyQt import QtWebKit
from PyQt.QtWebKit import *

def main():
        print 'Hola mundo'
        a = 5
        subrutina()
        print(a)
        return

def subrutina():
        a = 2
        print(a)
        return

if __name__ == 'main':
        main()
:::
```

En la figura 41 se muestra el código resaltado resultante que se obtiene en el pre visualizado:

```
import sys
import os
from PyQt import QtWebKit
from PyQt.QtWebKit import *

    def main():
        print 'Hola mundo'
        a = 5
        subrutina()
        print(a)
        return

    def subrutina():
    a = 2
    print(a)
    return


    if __name__ == 'main':
        main()
```

Fig. 41. Ejemplo de resaltado de código en lenguaje Python

- **Código en lenguaje Java**
  En el editor de textos se escribe:

  :::
  import java.io.IOException;
  import javax.servlet.*;

  public class Hola extends GenericServlet{

  public void service(ServletRequest request, ServletResponse response)

  throws ServletException, IOException
  {
  PrintWriter pw = response.getWriter();
  pw.println('HolaMundo');
  pw.close();
  }
  }
  :::

  En la figura 42 se muestra el código resaltado resultante que se obtiene en el pre visualizado:

```
import java.io.IOException;
import javax.servlet.*;
public class Hola extends GenericServlet
{
public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException
        {
            PrintWriter pw = response.getWriter();
            pw.println('HolaMundo');
            pw.close();
        }
}
```

Fig. 42. Ejemplo de resaltado de código en lenguaje Java

- **Código lenguaje C**
  En el editor de textos se escribe:

  :::
  #include
  int main(void) {
  int numero;

  while (i == 0)
  printf('Numero', numero);
  }
  if(condición 1){
  sentencia 1
  } else if(condición 2){
  sentencia 2
  }else if(condición n){
  sentencia n
  }else {
  sentencia por defecto
  }
  return 0;
  }
  :::

  En la figura 43 se muestra el código resaltado resultante que se obtiene en el pre visualizado:

```
#include

int main(void) {
    int numero;

    while (i == 0) {
          printf('Numero', numero);
    }

    if (condicion 1) {
        sentencia 1
     } else if (condicion 2){
     sentencia 2
           } else if (condicion n){
     sentencia n
           } else {
     sentencias por defecto

     return 0;
     }
```

Fig. 43. Ejemplo de resaltado de código en lenguaje #C.

7. Conclusiones

En el desarrollo de este trabajo terminal se logró un procesador de textos sencillo que logra añadir elementos básicos para la creación, edición y modificación de un documento de una manera sencilla.

A lo largo de este trabajo se presentaron contratiempos a la hora del diseño la gramática, estuvieron presentes algunos conflictos de reducción de reglas gramaticales, ya que algunos de los elementos que se implementaron tiene una sintaxis muy parecida entre sí, estos conflictos fueron resueltos principalmente de dos maneras; la primera de ellas fue con la ayuda de la biblioteca PL, la cual prioriza el orden en el cual las reglas gramaticales son definidas, la segunda manera fue redefiniendo algunos de los tokens y de las reglas gramaticales que se habían considerado en un inicio.

A partir de la gramática implementada es posible ampliar la sintaxis para proporcionar al usuario una mayor cantidad de elementos y símbolos para la creación y modificación de documentos y que estos sean más complejos y/o robustos.

La herramienta PLY facilita la creación de un compilador tanto en el análisis léxico como el análisis sintáctico ya que provee de funciones como ambigüedad en la gramática y manejo de errores.

Se pudo comprobar que es factible simplificar la escritura de documentos por medio de comandos reduciendo la sintaxis usada para algunos elementos en el procesador de redacción, facilitando al

usuario la creación de estos.

El paradigma ‘lo que ves es lo que quieres decir’ donde existe la separación entre contenido y presentación presenta ventajas a la hora redactar documentos ya que permite al usuario principalmente enfocarse en el contenido del documento y deja en mayor medida el formato y estilos al sistemas de exportación al procesador de textos.

8. Trabajo a futuro

El desarrollo de este procesador de textos proporciona una gran cantidad de posible trabajo a futuro ya que se puede extender hacia diferentes ejes de investigación y desarrollo.

Los principales puntos en los cuales se puede extender son:

- Ampliar la gramática para que acepte un mayor número elementos para la edición de documentos.
- Agregar otras funciones a la interfaz de usuario para que esta sea más completa, como incluir diagramas, permitir tablas de contenido, cambiar tipos de letra, entre otras.
- Desarrollar algún algoritmo de colaboración para que el documento se pueda editar de forma compartida entre uno o varios usuarios.
- Proveer al usuario de una mayor cantidad estilos para dar formato al documento.

9. Anexos

A continuación se muestran aspectos importantes a la hora del desarrollo de la aplicación

9.1. Gramática

Se expresan cada una de las reglas gramaticales que se han definido para el análisis sintáctico del procesador de textos con la notación de Backus-Naur.

<documento> ::= <lineas>

<lineas> ::= <lineas> | <linea>

<linea> ::= <elementos>

<elementos> ::= <elementos> | <elemento>

<elemento> ::= <cita> | <fecuacion> | <parrafos> | <ccodigo> | <negritas> | <cursiva> | <subrayado> | <enfasis> | <stextos>| <titulo> | < separador> | <enlace> | <ulist> | <nlist> | <fecha> | <autor>

<titulo> ::= <H1> <stextos> <H1>
<titulo> ::= <H2> <stextos> <H2>
<titulo> ::= <H3> <stextos> <H3>
<titulo> ::= <H4> <stextos> <H4>
<titulo> ::= <H5> <stextos> <H5>
<titulo> ::= <H6> <stextos> <H6>

<fecuacion> ::= <SIGNOPESO> <SIGNOPESO> <ecuaciones> <SIGNOPESO><SIGNOPESO>

<ecuaciones> ::= <ecuaciones> | <ecuacion>

<ecuación> ::= <espacios> | <enters> | <palabras> | <EX> <MEN> | <LIN> | <PER> | <PLUS> | <IGUAL> <COMA> | <SLASH> | <PUNTO> | <AMPER> | <UNDER> | <BSLASH> | <BBSLASH> | <IPAREN> | <DPAREN> | <ILLAVE> | <DLLAVE> | <NUMERO> | <ICORCHETE> | <DCORCHETE> | <SEMICOLON> | <CIRCUMFLEX> | <APOSTROPHE> |

	<ASTERISCO>
<parrafos>	::= <parrafos> <parrafo>
<parrafo>	::= <autor> | <espacios> | <stextos> | <enlace> | <negrita> | <cursiva> | <cnegrita> | <subrayado>
<negritas>	::= <ASTERISCO> <stextos> <ASTERISCO>
<subrayado>	::= <GUIONBAJO> <stextos> <GUIONBAJO>
<cursiva>	::= <ASTERISCO> <ASTERISCO> <stextos> <ASTERISCO> <ASTERISCO>
<énfasis>	::= <ASTERISCO> <ASTERISCO> <ASTERISCO> <stextos> <ASTERISCO> <ASTERISCO> <ASTERISCO>
<cita>	::= <MAYORQUE> <ESPACIO> <selemento> <ENTER>
<cita>	::= <MAYORQUE> <parrafo> <ENTER>
<enlace>	::= <ICORCHETE>< stextos> <DCORCHETE> <IPAREN> < url> <IDPAREN>
<imagen>	::= <EX> < ICORCHETE> <stextos> <DCORCHETE> <IPAREN> <locaciones> < DPAREN>
<imagen>	::= <EX> < ICORCHETE> <stextos> <DCORCHETE> <IPAREN> <locaciones> <DPAREN> <IPAREN> <NUMERO> <COMA> <NUMERO> <DPAREN>
<selementos>	::= <selementos>< selemento>
<selemento>	::= <negrita> | <cita> | <stextos> | <cursiva>
<palabras>	::= <AACEN> | <EACEN> | <IACEN> | <OACEN> <UACEN> | <NACEN> |<PALABRA>
<espacios>	::= <espacios> | <ESPACIO>

<enters> ::= <enters> | <ENTER>

<ulist> ::= <uitems> | <uitem>

<uitem> ::= <ASTERISCO> <ESPACIO> <palabras> <ENTER>

<uitem ::= <ASTERISCO> <palabras> <ENTER>

<nlist> ::= <nitems> | < nitem>

<nitem> ::= <NUMERO> <PUNTO> <ESPACIO> <palabras> <ENTER>

<nitem> ::= <NUMERO> <PUNTO> <palabras> <ENTER>

<fecha> ::= <IGUAL> <FECHA>

<separador> ::= <DMIN> <enters>

<separador> ::= <DMIN> <espacios> <enters>

<separador> ::= <DMIN> <IPAREN> <NUMERO> <DPAREN> <enters>

<autor> := <IGUAL> <AUTOR> <ILLAVE> <stextos> <DLLAVE>

<stextos> ::= <stextos> <stexto>

<stexto> ::= <espacios> | <enters> | <palabras> | <EX> | <MEN> | <AST> | <MENORQUE> | <MAYORQUE> | <GATO> | <DOLLAR> | <PER> | <PLUS> | <MARK> | <SLASH> | <DSLASH> | <AMPER> | <COMA> | <GUIONBAJO> | <PUNTO> | <ILLAVE> | <NUMERO> | <IPAREN> | <COMILLA> | <ICORCHETE> | <DCORCHETE> | <SEMICOLON> | <CIRCUMFLEX> | <APOSTROPHE'>

<locaciones> ::= <locaciones> |< locacion>

<locacion> ::= <palabras' <JPG> | <PNG> | <MEN> | <SLASH> | <PUNTO>

<url> ::= <HTTP> < COLON> <DSLASH> <dirs'>

<dirs>	::= <dirs> | <dir>
<dir>	::= <palabras> | <H1> | <PER> | <WWW> | <MEN> <PLUS> | <MARK> | <IGUAL> | <AMPER> | <SLASH> | <UNDER> | <PUNTO>
<ccodigos>	::= <COLON> <COLON> <COLON> <códigos> <COLON> <COLON> <COLON>
<codigos>	::= <codigos> | <codigo>
<codigo>	::= <TAB> | <espacio> | <palabras> | <UNDER> | <PRINT> | <PRINTF> | < RETURN> | <IF> | <ELSE> | <FOR> | <CLASS> | <IMPORT> | <THROWS> | <PRINTLN> | <FROM> | <INT> | <ENUM> | <VOID> | <PUBLIC> | <EXTENDS> | < LONG> | <WHILE> | <SWITCH> | <DEF> | <CASE> | <TYPEDEF> | <GETWRITER> | <CHAR> | <CONTINUE> | <DO> | <SIZEOF> | <COLON> | <IPAREN> | <DLLAVE> | <ICORCHETE> | <DCORCHETE> | <IGUAL> | <COMA> | <PUNTO> | <MARK> | <SEMICOLON> | <MARK> <ASTERISCO> <PLUS> | <NUMERO>
<codigo>	::= <MEN> <MAYORQUE> <PER>
<codigo> <APOSTROPHE>	::= <APOSTROPHE> <palabras>
<codigo>	::= <APOSTROPHE> <stextos> <APOSTROPHE>
<codigo>	::= <COMILLA> <stextos> <COMILLA>

9.2. Requerimientos

Para el uso de la aplicación es necesario tener consideradas los siguientes aspectos:

- Instalación del lenguaje de programación Python
- Instalación de la biblioteca PLY
- Espacio en disco de 26MB

10. Bibliografía

[1] 20000268, “Locución de textos con animación”, Dulce Angélica Ruz Rodríguez, Israel Damián Castillo de la Paz

[2] 2012B045 “Aplicación Móvil con Realidad Aumentada de Apoyo a Turistas en Traducción de Textos”, Alan Omar Junio Martínez Bello, Jorge Emmanuel Morales Díaz, Carlos Daniel Rocha García

[3] Plantilla de preparación de artículos técnicos en procesador de texto Word [online]. México. Disponible en: http://ewh.ieee.org/reg/9/files/SAC/SPC2014/modelo_template_SPC_spanish.pdf

[4] Lyx, [online]. México. Disponible en: http://ewh.ieee.org/sb/el_salvador/uca/lyx.html

[5] Microsoft office, Word [En línea]. México. Disponible en: https://products.office.com/es-mx/word

[6] Microsoft office, Wordpad [En línea]. México. Disponible en:: http://windows.microsoft.com/es-mx/windows/using-wordpad#1TC=windows-7

[7] LibreOffice, Writer, [En línea]. México. Disponible en: https://www.openoffice.org/es/

[8] AbiWord, [online]. México. Disponible en: http://www.abiword.org/

[9] Latex – A document preparation system, latex [En línea]. México. Disponible en:https://www.latex-project.org/

[10] Markpad – Markpad Editor, latex [En línea]. México. Disponible en:http://code52.org/DownmarkerWPF/

[11] Procesadores de texto, En línea México Disponible en: http://assets.mheducation.es/bcv/guide/capitulo/8448169271.pdf

[12] Reporte Técnico, [En línea]. México. Disponible en: www.unsis.edu.mx/convocatorias/REPORTE_TECNICO.doc

[13] V.Aho, R. Sethi y D. Ullman, "Compiladores: Principios, técnicas y herramientas", PEARSON, 1990, pp. 820

[14] UML,”Uml Documentation”, [En línea]. México. Disponible en: http://www.uml.org/

[15] Pitón,"Python Documentation", [En línea]. México. Disponible en: https://www.python.org/.

[16] PLY,"PLY Documentación", [En línea]. México. Disponible en: http://www.dabeaz.com/ply/

[17] CSS, "HTMl en la web" [En línea]. http://www.acercadehtml.com/manual-html/que-es-html.html

[18] CSS, "Introduccion a CSS" [En línea]. Disponible en http://es.html.net/tutorials/css/lesson1.php

[19] Mathjax, "Mathjax, documentation" [En línea]. Disponible en: https://www.mathjax.org/

[20] Joe Di Castro, "Markdown & Pygments Lexers Cheat Sheet" [En línea]. Disponible en: http://joedicastro.com/pages/markdown.html

[21] Roger S. Pressman, "Ingeniería del Software: un enfoque práctico", Mc Graw Hill, Sexta edicion, 2011, 953