**Scanner**

**Definición general del Proyecto:**

Idea general: El Proyecto debe ser capaz de crear grafos de expresiones regulares para que el lexer sepa que aceptar y que no, el Proyecto debe tirar como salida una linkedlist con tokens válidos o mostrar un reporte de errores.

Objetivos: El objetivo principal es que la persona pueda definir su lenguaje de programación desde el input y luego pueda probarlo.

Usuarios: Esto está dirigido a programadores.

**Requerimientos del software**

Requisitos generales:

* El proyecto debe ser capaz de aceptar cualquier expresión regular
* El proyecto debe de ser capaz de mostrar errores
* El proyecto debe de ser capaz de tokenizar incluso si se encuentran errores y dar un reporte detallado de ambos
* El proyecto a su vez debe ser manejable por líneas de comando

Requisitos funcionales

* El proyecto es capaz de aceptar cualquier expresión regular.
* El proyecto crea grafos a partir de las expresiones regulares
* Se puede escribir líneas de programación en un txt y el programa genera errores o éxitos.
* El programa muestra la información de una manera ordenada
* El programa muestra la información de errores y brinda un reporte de ello
* El programa es manejable desde la terminal por comandos.

Información de autoría

* El proyecto es 100% original, se usaron algunos recursos para definir ciertas partes del código, pero como tal el código fue diseñado por nosotros.

Alcances del sistema:

* La única limitación de este software es el tema de \n, \t, \\*, etc., dentro del string o char dado que no sabemos cómo manejar eso dentro del string o char, aparte de ello nuestro sistema alcanza todos los demás requisitos solicitados anteriormente

**Especificaciones de procedimiento**

* Procedimiento de desarrollo: Para este proyecto utilizamos visual studio code para poder programar nuestro lexer a su vez utilizamos herramientas como las librerías dentro de visual studio code y utilizamos stackoverflow como referencia a dudas que surgían en el camino.
* Planificación: La forma como planificamos este proyecto fue que íbamos a ir trabajando por iteraciones entonces nos guiamos por la guía de miu y la primera semana fue dedicada a investigación y desarrollo de los nfa, la segunda semana fue dedica al desarrollo de los nfa a dfa y exportación y la última semana fue dedicada al lexer como tal y documentación.

**Procedimiento de instalación y prueba**

* Requisitos no funcionales: Como hemos detallado, la única restricción que limita nuestro programa es el tema de \n, \t, etc. Dentro de un string o char.
* Obtención e instalación: Para instalar nuestro software debes ir al siguiente link: <https://github.com/pjvl99/compilador>, una vez ahí le das en el botón verde de “code” y lo descargas como zip, para instalarlo se hace desde la terminal, se compila el principal “Compiler.cpp” con: g++ Compiler.cpp -o, esto creara un ejecutable, una vez hecho esto con los comandos –target, -debug puedes compilar lo demás, si en dado caso tuvieras duda con los comandos solo pon: “./a.exe” y esto te mostrara la descripción de los comandos, recuerda siempre estar ubicando en la carpeta “Compiler” del proyecto para que te funcione, en la misma encontraras los .txt, el de regex sirve para definir las expresiones regulares, en la primera línea defines el nombre de esa expresión y en la segunda línea la expresión regular como tal, se vuelve a repetir con la 3ra, 4ta etc. No olvides poner “END” para que el programa sepa hasta donde leer, luego el otro .txt editable es el de “código” ahí puedes escribir tu código para probar.
* Especificaciones de prueba y ejecución: Para correr el programa basta con tener instalado algún compilador de C/C++ o usar directamente visual studio code con la librería C/C++ (recomendado), no olvides que puedes correr directamente desde visual studio code o desde la terminal, solo debes tener instalado g++.

**Arquitectura del sistema**

* Descripción jerárquica: El programa esta ordenado por una clase padre llamada “Compiler” que controla las demás, en este caso debajo de esta están Scanner y Lexer.
* Diagrama de módulos:

Compiler

Input Codigo.txt

Input regex.txtzx

Tokens (Linkedlist)

Lexer

Grafos.txt

Scanner

**Descripción de módulos**

* Compiler: Este módulo tiene por objetivo manejar los demás módulos dentro del compilador, este módulo depende del input que el usuario ingrese en la terminal, este se encuentra en la carpeta de Compiler, a su vez todos los archivos en cada carpeta se encuentran implementados en el módulo.
* Input regex: Este módulo simplemente sirve de input, controlado por la librería “iostream” de C/C++, aquí el usuario define las expresiones regulares que el scanner pasara a interpretar.
* Grafos: Este archivo es el output de Scanner, aca están los grafos codificados para el lexer.
* Lexer: Acá se recibe por input los grafos y el código.txt donde simplemente en base a los grafos definidos así tokenizara o tirara error.
* Código: En este archivo el usuario simplemente escribe lo que quiere programar.
* Scanner: Aca simplemente se interpretan las expresiones regulares y se exporta a un .txt los grafos creados.
* Tokens: En el último modulo simplemente se muestran errores y tokens.

**Funciones:**

* Compiler:
  + Main(): Se controla todo el compilador
* Scanner:
  + Scanner(): Corazon del programa, aquí se controla la creación y exportación de grafos
  + Casos(): Se analiza carácter por carácter del regex para discriminar y crear nfa
  + Parentesis(): Se analizan paréntesis para anticiparse a eventualidades
  + Asterisco(): Hace la función de \* o ? según sea necesario.
  + Transform(): Recibe el nfa y lo transforma a dfa y exporta.
* Lexer:
  + Lexer(): Controla el tema de tokenizacion y discriminación de input.
  + Crear(): Recibe los grafos del scanner y los guarda en variables.
  + Punteros(): Con los punteros se verifica información en base al input de código para mostrar errores o tokenizar.

**Conclusiones**

Honestamente nos parece un gran proyecto, el tema de lo que hace un analizador léxico es algo sumamente increíble, sin embargo, cabe destacar que ha sido el proyecto que más nos ha costado hasta ahora en la carrera y al inicio no estuvimos muy orientados sobre qué hacer o cómo hacerlo.

Sin embargo, con un par de leídas a los libros nos orientamos y pese que estuvimos casi 20 días luchando porque funcione, funciona a un 90% y estamos orgullosos de ello, podemos concluir que ha sido una experiencia enriquecedora y estamos preparándonos mentalmente para la fase 2.

Por ultimo podemos concluir la importancia de un analizador léxico no solo para un compilador sino para otras cosas como traducción de lenguajes o códigos de otra índole, es algo trabajoso, cansado y hasta cierto punto frustrante dada la dificultad que lo acompaña, pero vale la pena.