Ejercicio 1

1. Funcion `retractar`

Maneja el puntero forward, y begin lexema, cuando falla vuelve retrocede el puntero (f) y adelanta a la posision de f el puntero lb

- **En que momento se invoca esta funcion** Se invoca cuando el autómata alcanza un estado de aceptación que requiere haber leido el siguiente carácter para confirmar el fin del token actual.
- **Que sucede con el puntero del buffer al ejecutarse** Al ejecutarse, el puntero de lectura del bufer retrocede una o más posiciones.
- **Por que es necesaria para aceptar correctamente un token.** Es necesaria porque, a menudo, la unica forma de saber que un token ha terminado es leyendo el siguiente caracter que no pertenece al token actual.
- **Como influye en el comportamiento del automata finito:** Permite que el automata sea mas flexible con la entrada, asegurando que cada token comience exactamente donde terminó el anterior, pudiendo reconocer varios tokens en una lista sin que un caracter no reconocido rompa todo.

2. Función `fallo`

Gestiona las situaciones donde el flujo de caracteres no coincide con ninguna siguiente posiscion en un estado.

- **Que significa que un automata "falle"** Significa que el automata ha llegado a un estado donde no existe una transicion posible para el simbolo actual de forward.
- **Que ocurre cuando no existe una transicion valida para el caracter leido.** Cuando no hay transición valida, el proceso de reconocimiento actual se detiene y se busca una alternativa.
- **Como permite esta funcion intentar el reconocimiento de otro patron** Esta funcion permite reiniciar el análisis desde el último punto con transicion valida y estado match, probando con el siguiente automata o expresión regular.
- **Su relacion con la estrategia de maxima coincidencia** Si una ruta falla despues de haber pasado por un estado de aceptacion previo, el sistema puede fallar hacia atras para aceptar la cadena más larga encontrada anteriormente.

3. Función `error`

Define el comportamiento del sistema con secuencias de caracteres no validas.

- **En que situacion se produce un error lexico.** Se produce cuando ya no hay ninguna trancision vailida en todo el automata para el carater leido.
- **Que ocurre cuando ningun automata reconoce la secuencia de entrada** Cuando ningun automata reconoce la secuencia, el analizador lanza error ya que no puede hacer nada, solo podria dar sujerencias, o entrar en modo panico.
- **Que mecanismos de recuperacion pueden aplicarse** Reporta indicando la línea, y el caracter o secuencia de caracteres que causaron el conflicto.
  - **Modo Panico:** Ignorar caracteres hasta encontrar un delimitador
  - **Modo panico2:** busca siguiente carater que sea detecable por la maquina.
  - **Reparación de cadena:** Intentar insertar, borrar o sustituir un caracter para que la entrada sea válida.

4. Ejemplo de Aplicación

![[Pasted image 20260212211150.png]]

---

7. Cambiar el color de palabras al estilo VS Code

VS Code usa archivos de gramática TextMate (`.tmLanguage.json`) para colorear codigo. Cada regla tiene un patron regex y un "scope" que indica que tipo de token es (keyword, string, comment, etc.), y el tema de colores se encarga de asignarle un color a cada scope.

- **Que habria que hacer** Se tendria que crear una extension de VS Code que incluya un archivo de gramatica con los mismos patrones regex que se definen en el lexer, pero en formato JSON. Cada patron lleva un nombre de scope estandar como `keyword.control` o `string.quoted.double`.
- **Como se conecta con el analisis lexico** Es esencialmente el mismo trabajo que hacer un lexer: los patrones son las mismas expresiones regulares, y los tipos de token se convierten en scopes. La diferencia es que el resultado se usa para pintar en lugar de para parsear.
- **Como aplica los colores el tema** Al usar nombres de scope convencionales, cualquier tema instalado aplica sus colores automaticamente sin configuracion extra, porque todos los temas mapean los mismos scopes estandar.

---

8. Herramienta similar a Flex: ANTLR4

ANTLR (ANother Tool for Language Recognition) es un generador de analizadores creado por Terence Parr. A diferencia de Flex que solo genera lexers en C, ANTLR genera tanto el lexer como el parser a partir de un unico archivo de gramatica `.g4`, y soporta multiples lenguajes de salida (Java, Python, Go, C#, JavaScript, entre otros).

- **Como funciona** Se escribe una gramatica `.g4` donde las reglas en MAYUSCULAS son las reglas lexicas (equivalentes a los patrones de Flex) y las reglas en minusculas son las reglas sintacticas. ANTLR genera el codigo fuente del lexer y parser automaticamente.
- **Diferencia principal con Flex** Flex solo reconoce tokens; ANTLR ademas construye el arbol de parseo completo y permite recorrerlo con patrones visitor o listener. Tambien resuelve ambiguedades automaticamente en lugar de depender del orden de las reglas.
- **Por que es relevante** Es la herramienta estandar en muchos cursos de compiladores y se usa en proyectos reales como la gramatica oficial de Python y el compilador de Kotlin.

Link repo https://github.com/Jonialen/compis1-lab1.git