Almacenamiento en búfer de entrada

Durante el análisis, el lexer necesita leer **más caracteres** de los que forman un lexema para saber dónde termina.
Por ejemplo, al leer > se debe verificar si el siguiente carácter es =, formando >=.
A veces es necesario **devolver** los caracteres leídos de más para procesarlos en el siguiente token.

Pares de búferes

Una técnica común es usar **dos búferes** grandes, de manera que mientras uno se procesa, el otro se llena con más texto, evitando interrupciones en la lectura.

Centinelas

Los **centinelas** son caracteres especiales que indican el final de un búfer, facilitando el control de lectura sin necesidad de verificar constantemente los límites del archivo.

Especificación de los tokens

Cadenas y lenguajes

Antes de poder definir los tokens de un lenguaje de programación, es necesario comprender algunos conceptos fundamentales.

Un **alfabeto** es un conjunto finito de símbolos, por ejemplo, el conjunto de letras, dígitos o signos que puede usar un lenguaje.
Una **cadena** es una secuencia finita de símbolos de ese alfabeto. También se le llama **palabra** u **oración**.
La **longitud de una cadena** es el número de símbolos que contiene, incluyendo los repetidos.
Un **lenguaje** es un conjunto de cadenas formadas con los símbolos del alfabeto.

Cuando unimos dos cadenas, realizamos una **concatenación**, que consiste en colocar una cadena inmediatamente después de otra. Si tenemos las cadenas *s* y *t*, su concatenación se escribe como *st*.

Otras definiciones importantes son:

- Un **prefijo** de una cadena *s* se obtiene eliminando cero o más caracteres del final de *s*.
- Un **sufijo** se obtiene eliminando cero o más caracteres del inicio.
- Una **subcadena** se obtiene eliminando tanto un prefijo como un sufijo.
- Una **subsecuencia** se forma eliminando símbolos en posiciones específicas de la cadena, sin importar si están contiguos.

Los **prefijos, sufijos y subcadenas propias** son aquellos que no incluyen la cadena vacía (ε) ni la cadena completa *s*.

Operaciones sobre lenguajes

Sea L y M dos lenguajes sobre el mismo alfabeto. Existen varias operaciones entre ellos:

- **Unión (L ∪ M):** contiene todas las cadenas que están en L, en M o en ambos.
- **Concatenación (LM):** consiste en todas las cadenas que pueden formarse concatenando una cadena de L con una de M.

También pueden definirse **potencias de un lenguaje**:

- $L^0 = \{\varepsilon\}$, el conjunto que contiene solo la cadena vacía.
- $L^{i+1} = L^i L$.

A partir de esto se definen:

- **Cerradura de Kleene (L*):** unión de todas las potencias de L, es decir, $L^0 \cup L^1 \cup L^2 \cup \ldots$.
- **Cerradura positiva (L^+):** unión de todas las potencias excepto la cadena vacía, $L^1 \cup L^2 \cup \ldots$.

Las dos cerraduras se relacionan así:

- $L^* = \{\varepsilon\} \cup L^+$
- $L^+ = L\,L^*$

Expresiones regulares

Las **expresiones regulares (ER)** son una forma compacta y formal de describir los lenguajes. Se construyen usando:

- los símbolos del alfabeto,
- la unión (|),
- la concatenación, y
- el operador de cierre (*).

Definición formal:

1. **ε** representa la cadena vacía, con lenguaje asociado $L(\varepsilon) = \{\varepsilon\}$.
2. Cada **símbolo x** del alfabeto representa el lenguaje $L(x) = \{x\}$.
3. **Concatenación (rs):** $L(rs) = L(r)L(s)$.
4. **Unión (r|s):** $L(r|s) = L(r) \cup L(s)$.
5. **Cerradura (r*):** $L(r^*) = (L(r))^*$.
6. **Agrupación (r):** $L((r)) = L(r)$.

Precedencia de operadores:

1. El operador de cierre (*) tiene la prioridad más alta.
2. La concatenación tiene prioridad intermedia y es **asociativa a la izquierda**.
3. La unión (|) tiene la prioridad más baja y también es **asociativa a la izquierda**.

Además, el cierre positivo puede expresarse como $r^+ = r\,r^*$.

Definiciones regulares

Las **definiciones regulares** son similares a las producciones de una gramática libre de contexto, pero menos poderosas.
Si Σ es un alfabeto, una definición regular tiene la forma:

```
d1 → r1
d2 → r2
...
dn → rn
```

Donde cada d_i es un nombre único (no pertenece al alfabeto Σ), y r_i es una expresión regular que puede usar los símbolos del alfabeto y las definiciones anteriores.

Es importante notar que:

- Cada definición puede depender solo de las anteriores (d_1 a d_{i-1}).
- No puede depender de definiciones posteriores.

Estas definiciones no amplían el poder expresivo de las expresiones regulares, solo facilitan su escritura y comprensión.

Extensiones de las expresiones regulares

Existen algunas **extensiones útiles** para simplificar las expresiones:

1. **Cierre positivo (+):** representa una o más repeticiones ($r^+ = r\,r^*$).
2. **Cero o una ocurrencia (?):** significa que el elemento puede aparecer una vez o no aparecer ($r? = r \mid \varepsilon$).
3. **Clases de caracteres ([ ]):** agrupan varios símbolos.
    - Por ejemplo, [abc] equivale a $a \mid b \mid c$.
    - Si los símbolos son consecutivos, se puede abreviar como [a-c].

Estas extensiones son solo **atajos sintácticos**; no agregan nuevos tipos de lenguajes.

Ejemplo: Constantes flotantes

Una **constante flotante** se forma combinando partes numéricas y opcionales:

- una parte entera y una fraccional opcional (ambas son secuencias de dígitos),
- un punto decimal opcional después de la parte entera,
- un exponente opcional (precedido por *e* o *E*, con signo opcional),
- y un sufijo opcional (*f, F, l o L*).

El tipo de la constante depende del sufijo:

- *f* o *F* → flotante,
- *l* o *L* → doble largo,
- sin sufijo → doble.

digito → [0-9]

letra → [a-zA-Z]

digitos → digito +

fraccion_opt → (. digitos) ?

exponente_opt→ (E (+|-)? digitos)?

sufijo_opt → ((f|F)|(l|L))?

num → dígitos fracción_opt exponente_opt sufijo_opt