72.39 - Autómatas, Teoría de Lenguajes y Compiladores

Proyecto CONCAT

Integrante:

- Scheffer, Tomás Guillermo (63393)

Fecha:

- 24 de junio de 2025

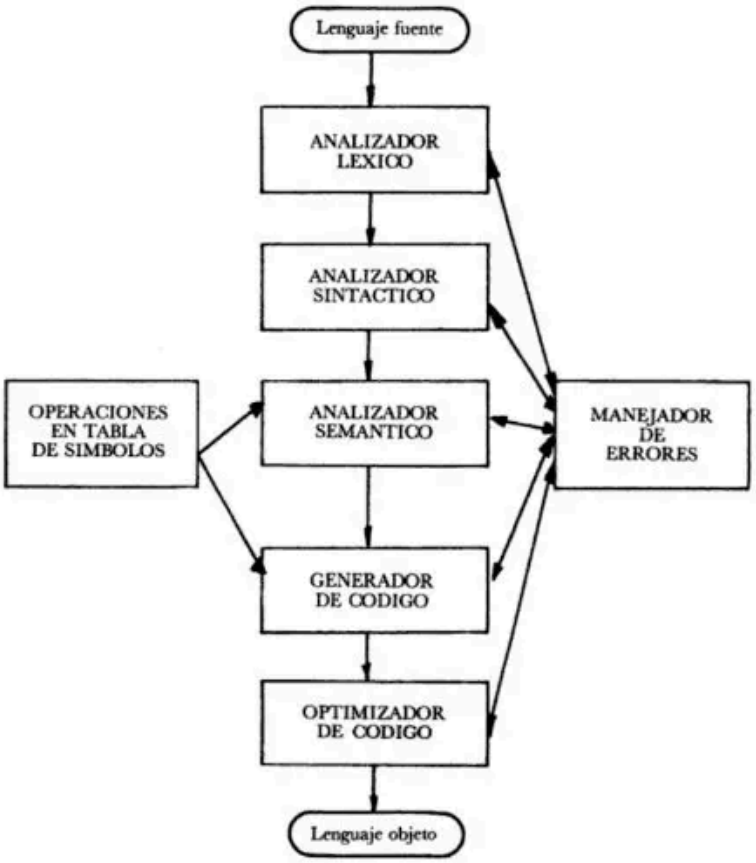

Tabla de contenidos

Introducción

Este compilador nació de la idea de facilitar la generación de cadenas de texto codificadas en ASCII orientado principalmente a tareas de transformación, manipulación y presentación de texto. El objetivo estuvo en el diseño de un lenguaje de dominio específico que simplifique operaciones comunes sobre strings como transformaciones (ej. pasar de minúscula a mayúscula), realizar reemplazos, cifrar/encriptar, generación de cadenas aleatorias y posteriormente interpolación de variables.

No se conocía en principio el alcance total del lenguaje por lo que este mismo sufrió alteraciones a lo largo de su desarrollo, pero siempre conservando la idea principal de facilitar estas transformaciones sobre texto plano. Asimismo se cree que este lenguaje no está cerrado, con eso quiero decir que por la naturaleza del objetivo siempre existirán oportunidades para continuar con su evolución.

Consideraciones Adicionales

Como se ha dicho en la introducción por la naturaleza del lenguaje siempre tendrá oportunidad de mejora y es por esto que en un principio se trató de conservar una arquitectura modular y flexible que permitiera la mayor posibilidad de expansión y/o mejora del lenguaje. Con esto en mente no se cerró la posibilidad de realizar más tipos de operaciones, tipos de variables y capacidad de expresión en general. Esto puede ser notado por la gramática del lenguaje. Esta ha sido diseñada para aceptar construcciones que aún no están completamente soportadas por el generador de código actual. Esto no es en sí un error ni una omisión, sino una elección orientada a anticipar futuras extensiones del lenguaje y simplemente para permitir la realización del funcionamiento correcto de la herramienta sin restricciones duras.

Desarrollo del proyecto

- Definición del lenguaje

El lenguaje está fuertemente inspirado por C ya que es un lenguaje extremadamente conocido y simple de usar. Pero a diferencia de este, CONCAT ofrece una semántica centrada exclusivamente en operaciones textuales.

A nivel sintáctico el lenguaje permite la declaración de variables tipo String y Atomic (equivalente a integer en C), asignaciones mediante expresiones, ejecución de rutinas y producción de salida a consola mediante la instrucción 'OUT'.

A nivel expresivo se incorporan funciones integradas para la realización de operaciones sobre texto, estas son:

- REV("Texto") -> invierte el orden de los caracteres.
- TUP("Texto") -> convierte todo el texto a mayúsculas.
- TLO("Texto") -> convierte todo el texto a minúsculas.
- LEN("Texto") -> devuelve la longitud del texto.
- RPL("Chau mundo", "Chau", "Hola") -> reemplaza todas las palabras que aparezcan en el segundo argumento por la tercera sobre el texto en el primer argumento.
- RND(5, 10, "abc") - genera una cadena de texto aleatoria de entre 5 y 10 caracteres usando solo los caracteres presentes en el tercer argumento.
- ECP("mensaje", "clave") -> encripta el primer argumento con operación XOR contra el segundo argumento, y se codifica a base64.

Todos estas operaciones devuelven un String y permiten el anidamiento.
Además se permite la interpolación de variables declaradas sobre Strings usando la notación "${variable}". Esto facilita la construcción de mensajes dinámicos.

Ejemplo:

```
String nombre = “Juan”;
OUT(“Hola ${nombre}, bienvenido.”);
```

Salida: “*Hola Juan, bienvenido.*”

La interpolación también admite variables de tipo Atomic, y se pueden usar cuantas se quiera sin limitación de cantidad.

También se definió la manera de generar Rutinas mediante la palabra clave “FUN” permitiendo agrupar instrucciones mediante un identificador. Estas rutinas actualmente no aceptan parámetros ni devuelven valores, pero permiten agrupación de código y encapsulación de lógica. Para realizar una invocación solo es necesario su nombre con un signo de exclamación final (“!”).

Ejemplo:

```
String nombre = “Juan”;

FUN rutina {
        OUT(“Hola ${nombre}, bienvenido.”);
}

rutina!;
```

Salida: “*Hola Juan, bienvenido.*”

Una característica principal de este lenguaje es la capacidad de composición, ya que cualquier función integrada puede recibir como argumento otra función integrada.

Ejemplo:

```
OUT(TUP(REV(“hola”)));
```

Salida: “ALOH”

Ejemplo de programa y explicación:

```
>> String nombre = "juan";
>> String saludo = "Hola ${nombre}!";
>>
>> String texto = "Esto es un texto de prueba que voy a modificar";
>> String textoA = TUP(texto);
>> String textoB = RPL(textoA, "PRUEBA", "experimento");
>> String textoC = REV(textoB);
>>
>> Atomic numero = 123;
>> String clave = "clave${numero}";
>> String mensaje = ECP(textoC, clave);
>>
>> OUT("Mensaje sin encriptar: ${textoC}");
>> OUT("Mensaje encriptado: ${mensaje}");
>>
>> FUN generar {
>>    String aleatorio = RND(8, 12, "abc123456");
>>    String longitud = LEN(aleatorio);
>>    OUT("Aleatorio (${longitud}): ${aleatorio}");
>> };
>> generar!;
>> generar!;
```

Salida:

```
Mensaje sin encriptar: RACIFIDOM A YOV EUQ otnemirepxe ED OTXET NU SE OTSE
Mensaje encriptado: MS0iPyN4dnw=
Aleatorio (9): 6526b1a5c
Aleatorio (8): c5453561
```

Explicación:

Este programa ejemplifica un poco mejor la capacidad actual del lenguaje CONCAT. Comenzamos con la declaración de una variable ‘nombre’ que luego es

utilizada mediante interpolación en la variable ‘saludo’. Después se trabaja sobre un String más extenso llamado ‘texto’ al cual se le aplican diferentes transformaciones: primero se convierte a mayúsculas con el uso de “TUP”, luego se realiza un reemplazo de palabras utilizando “RPL” y finalmente se invierte con “REV”.
Posteriormente para definir a ‘mensaje’ se toma el texto transformado y se encripta con “ECP” la cual toma una clave que es producto de un texto e interpolación con un número. Después se imprime el texto transformado sin encriptar y el encriptado.
Por último se define la rutina ‘generar’ que genera una cadena aleatoria utilizando la función “RND”, calcula su longitud y muestra ambos valores mediante interpolación. Para probar justamente la aleatoriedad se llama a esta función dos veces, produciendo dos diferentes resultados.

- Front End

Durante la etapa de diseño del FrontEnd, uno de los principales desafíos fue definir el alcance exacto de la sintaxis del lenguaje y diferenciar claramente qué aspectos tenían validarse en la etapa sintáctica y cuáles debían postergarse a próximas etapas. Al comienzo del desarrollo, no era del todo clara, lo cual llevó a incorporar en la gramática ciertas restricciones que, en retrospectiva, podrían haberse resuelto más elegantemente desde el backend o el generador.
Un ejemplo concreto fue la decisión de permitir ciertas expresiones anidadas o interpolaciones dentro de cadenas, que si bien podían aceptarse desde la gramática de forma general, requerían una lógica más compleja para validar su validez semántica o su comportamiento esperado al momento de ejecutarlas.
Aun así, se optó por diseñar una gramática más rica de lo que el compilador soporta actualmente, anticipando futuras funcionalidades y evitando reescribir la sintaxis cuando se agreguen más capacidades expresivas.

- Back End

Desde un comienzo se trató de mantener una estructura modular como el front end. Se comenzó con la implementación del análisis semántico, junto a ella ya fue necesario implementar la tabla de símbolos que también fue posteriormente compartida con el Generator. Se trató de incorporar la mayor cantidad de validaciones posibles para garantizar el buen funcionamiento del compilador. También como fue costumbre con el front end, se agregaron Loggers para hacer seguimiento en todas las funciones. Esto fue fundamental para el desarrollo en sí

de todo el proyecto, ya que el funcionamiento es muy recursivo y no solo hay que saber dónde fallan las cosas sino cómo se fueron desencadenando los errores. La parte de debug fue sustancial en el desarrollo de este proyecto y tuvo mayor presencia en esta etapa. Por limitaciones de tiempo aquí es donde se decidió conservar aquellas funcionalidades más relevantes sin antes dejar abierta la posibilidad de expansión. También fue necesario a medida que se fueron probando/implementando las cosas hacer algunos cambios en etapas anteriores para obtener los resultados esperados.

- ## Dificultades encontradas

Todas las etapas tuvieron sus desafíos, pero algo que se dió en cada construcción de los módulos es la investigación previa. Esto tomó más tiempo del esperado y de hacerse otra vez se considera que es mejor "empezar tocando" y aprender como realizar las cosas mediante pruebas reales. También en retrospectiva podría haber sido mejor definir el lenguaje de forma más dura para la construcción de una gramática en principio más fácil de seguir. Todo tuvo su curva de aprendizaje pero a parte de las dificultades más obvias que aparecen en todo proyecto (y sobre todo con manejo de memoria), en este estuvo se cometió el error de tratar de partir con todo en lugar de en pasos incrementales.
También cabe destacar que como es de esperar igualmente, a medida que se aplica lógica para la resolución/generación de código esta se vuelve exponencialmente más difícil de seguir y es necesario analizar profundamente cada cambio.
Al margen de lo mencionado se considera que lo explicado por la cátedra, referencias y el ejemplo Calculator permitieron el correcto desarrollo de este proyecto aún ante la desafortunada pérdida de integrantes originales del grupo de desarrollo.

- ## Posibles mejoras y/o expansiones

Se mencionó brevemente al inicio de este informe pero CONCAT está pensado para soportar mejoras de forma continua gracias a su gramática flexible. Existen múltiples direcciones y funcionalidades posibles pero las más relevantes que se notan por el momento son estructura de control condicionales (if-else) y bucles como (for-while). También se quiso pero no terminó de implementar un tipo "Buffer" para el manejo de datos binarios que servirá de suplemento para por ejemplo el uso de algoritmos de encriptación mejores, lectura de archivos

externos y posible traducción a cadenas de texto. En cuanto a archivos respecta también sería de gran utilidad volcar el contenido del resultado de la compilación en un archivo con nombre pasado por parámetro (esto fue recomendado por la cátedra). Otra expansión sería el cálculo de expresiones numéricas y la capacidad de convertir implícita o explícitamente los 3 tipos básicos planteados para este lenguaje (Atomic, String, Buffer). Luego se podrían implementar mejoras en cuanto a la presentación del texto de salida y posibilidad de decidir Encoding (UTF-8, Unicode, Etc).

- Bibliografía

No se utilizaron recursos por fuera de lo provisto por la cátedra.