Importancia de los Compiladores, Analizador Léxico

Pedro O. Pérez M., PhD.

Desarrollo de aplicaciones avanzadas de ciencias computacionales
Módulo 3: Compiladores
Tecnológico de Monterrey

pperezm@tec.mx

05-2025

1. Capítulo 1
   - Procesadores de lenguajes
   - La estructura de un compilador
   - La ciencia de construir un compilador
   - Aplicaciones de la tecnología de los compiladores

2. Capítulo 3
   - El papel del analizador léxico
   - Implementación de un analizador léxico

¿Qué es un compilador?

Un compilador es una herramienta que traduce software escrito en un lenguaje a otro. Para realizar esta traducción, la herramienta debe ser capaz de comprender el lenguaje original (sintaxis, semántica). Además, necesita comprender el lenguaje final. Finalmente, debe tener un esquema que le permita hacer la traducción.

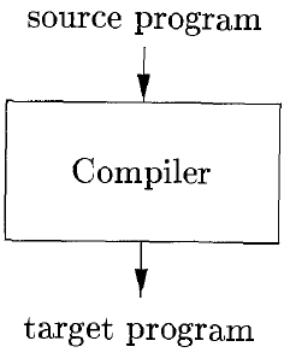


Figure 1.1: A compiler

¿Qué es un interprete?

Un interprete es otro tipo de herramienta común. En vez de producir un programa destino (o traducción), un interprete ejecuta directamente las operaciones especificadas en el programa fuente con las entradas dadas por el usuario.

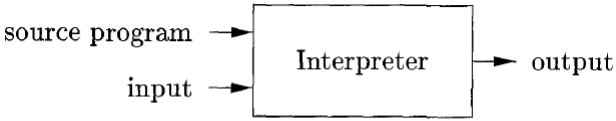


Figure 1.3: An interpreter

Un código destino producido por un compilador es usualmente más rápido que una código que está siendo interpretado. Sin embargo, un interprete puede, usualmente, dar mejor diagnóstico de errores que un compilador, porque ejecutar el código fuente instrucción por instrucción.

Pero, además tenemos esquemas híbridos...

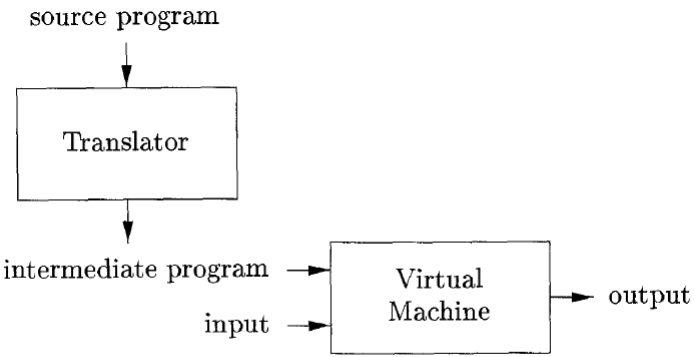


Figure 1.4: A hybrid compiler

¿Cómo es el proceso de generación de un código fuente?

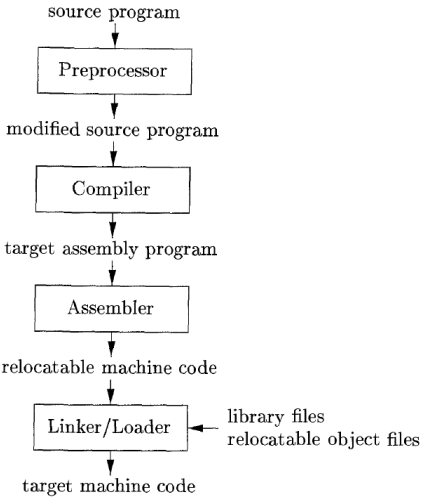
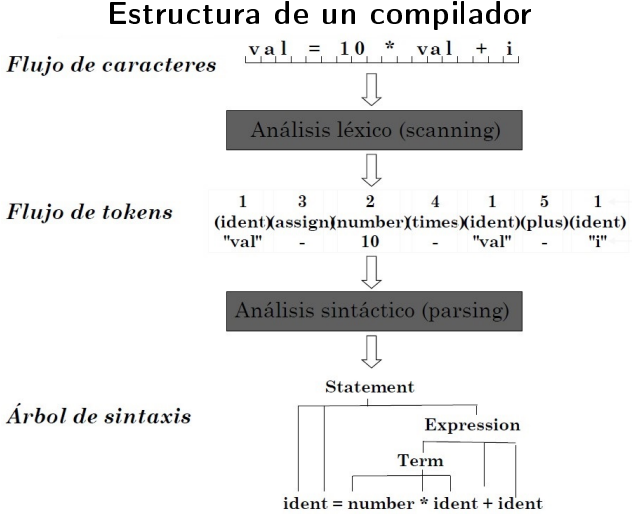
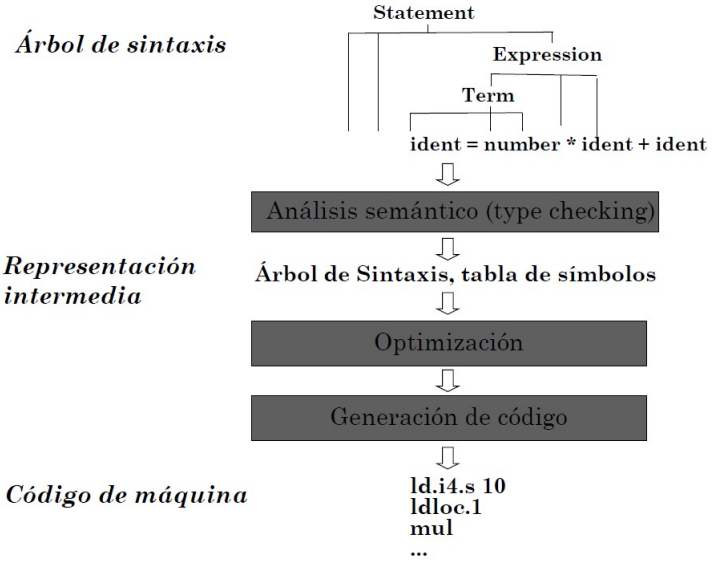


Figure 1.5: A language-processing system

La estructura de un compilador




Árbol de sintaxis
Statement
Expression
Term
ident = number * ident + ident
Análisis semántico (type checking)
Representación intermedia
Árbol de Sintaxis, tabla de símbolos
Optimización
Generación de código
Código de máquina
ld.i4.s 10
ldloc.1
mul
...

La ciencia de construir un compilador

Un buen compilador es un microcosmos del área de las ciencias computacionales: hace uso práctico de algoritmos codiciosos (asignación de recursos), técnicas de búsqueda heurística (programación de listas), algoritmos de grafos (eliminación de código muerto), programación dinámica (selección de instrucciones), autómatas finitos y push-down (escaneo y análisis), entre muchos otros temas. En otras palabras, trabajas en el diseño y programación de un compilador proporciona experiencia práctica en ingeniería de software que es muy difícil de obtener en sistemas más pequeños y menos complejos.

Aplicaciones de la tecnología de los compiladores

- Implementación de lenguajes de programación de alto nivel.
- Optimizaciones para arquitecturas computacionales:
  - Paralelismo a nivel de instrucciones.
  - Jerarquía de memorias.
- Diseño de nuevos arquitectura computacionales: Desde que los lenguajes de alto nivel son la norma, el desempeño de los sistemas computacionales está determinado no solo por su velocidad sino, también, por lo bien que los compiladores explotan sus características.
- Traducción de programas: traducción binaria (convertir código maquina a otro), hardware sintéctico (VHDL), interpretes de consultas a base de datos.

El papel del analizador léxico

- La tarea principal de un analizador léxico es leer los caracteres de entrada de un programa fuente, agruparlos en lexemas, y producir como salida una secuencia de tokens por cada lexema.

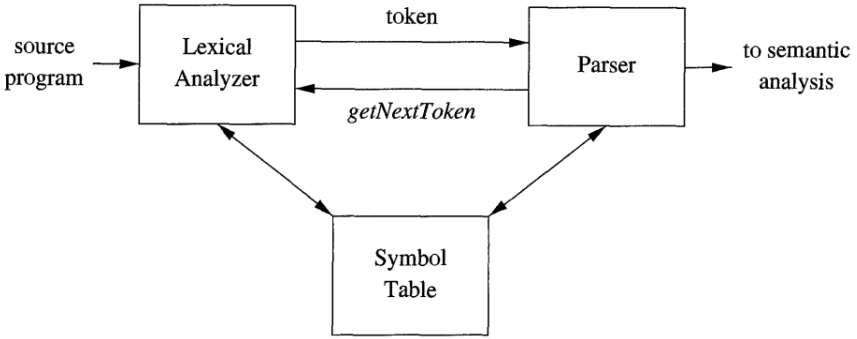


Figure 3.1: Interactions between the lexical analyzer and the parser

Análisis léxico vs. sintáctico

Existen tres razones principales por las cuales se separa el análisis léxico del sintáctico.

- Simplicidad del diseño.
- La eficiencia del compilador es mejorada.
- La portabilidad del compilador es aumentada.

Tokens, patrones y lexemas

- Un token es una dupla que consiste del nombre del token y un atributo opcional. El nombre del token es un símbolo abstracto que representa un TIPO de unidad léxica.
- Un patrón es una descripción de la forma que los lexemas de un token pueden tomar. En el caso de una palabra reservada, el patrón es la secuencia de caracteres que forman la palabra reservada.
- Un lexema es la secuencia específica de caracteres en el programa fuente que se emparejan con el patrón de un token determinado.

En la mayoría de los lenguajes de programación, las siguientes clases cubren la mayoría, sino todas, de tokens:

- Un token para cada palabra reservada.
- Token para operadores (individuales o grupales)
- Un token representando todos los identificadores.
- Uno o más tokens representan constantes (números o literales).
- Tokens para cada símbolo de puntuación.

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	< or > or <= or >= or == or !=	<=, !=
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but ", surrounded by "'s	`"core dumped"`

Figure 3.2: Examples of tokens

Errores léxicos

- Es difícil para un analizador léxico detectar un error en el programa fuente sin la ayuda de otros componentes.
- Sin embargo, supongamos que un analizador léxico es incapaz de continuar porque ninguno de los patrones de los token se emparejan con ningún prefijo de la cadena de entrada, ¿qué hacemos? Entramos en "modo pánico", ignora todos los caracteres de entrada hasta que pueda detectar un token bien formado.

Arquitectura de un analizador léxico basado en diagramas de transiciones

- Podemos organizar que los diagramas de transición para cada token sean tratados secuencialmente. De esta forma, la función `fail` reinicia el apuntador "`forward`" e inicia el siguiente diagrama de transición.
- Podemos ejecutar varios diagramas de transición "en paralelo", alimentado el siguiente carácter de entrada para todos ellos y permitiendo que cada uno haga la transición que se requiera (prefijo más largo).
- Podemos combinar todos los diagramas de transición en uno. De esta forma permitimos que el diagrama de transición lea la entrada y tome el lexema más largo que se empareje con un patrón.

```
TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character processing until a return
                  or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
```

Figure 3.18: Sketch of implementation of **relop** transition diagram

Implementando un analizador léxico

Revisión de la implementación de un analizador léxico:
`https://github.com/Manchas2k4/tc3002b`

Revisa la sección 3.5 de libro [AHO]