PROCESADORES DE LENGUAJE

Los lenguajes de programación son notaciones que describen los cálculos a las personas y las

máquinas. Nuestra percepción del mundo en que vivimos depende de los lenguajes de pro-
gramación, ya que todo el software que se ejecuta en todas las computadoras se escribió en

algún lenguaje de programación.

Pero antes de poder ejecutar un programa, primero debe tra-
ducirse a un formato en el que una computadora pueda ejecutarlo.

Los sistemas de software que se encargan de esta traducción se llaman compiladores. Este libro trata acerca de cómo diseñar e implementar compiladores. Aquí descubriremos
que podemos utilizar unas cuantas ideas básicas para construir traductores para una amplia
variedad de lenguajes y máquinas.

Además de los compiladores, los principios y las técnicas
para su diseño se pueden aplicar a tantos dominios distintos aparte, que es probable que un científico computacional las reutilice muchas veces en el transcurso de su carrera profesional.

El estudio de la escritura de los compiladores se relaciona con los lenguajes de programa-
ción, la arquitectura de las máquinas, la teoría de lenguajes, los algoritmos y la ingeniería de

software.

Un intérprete es otro tipo común de procesador de lenguaje. En vez de producir un progra-
ma destino como una traducción, el intérprete nos da la apariencia de ejecutar directamente

las operaciones especificadas en el programa de origen (fuente) con las entradas proporcionadas
por el usuario,

El programa destino en lenguaje máquina que produce un compilador es, por lo general, más
rápido que un intérprete al momento de asignar las entradas a las salidas. No obstante, por
lo regular, el intérprete puede ofrecer mejores diagnósticos de error que un compilador, ya que
ejecuta el programa fuente instrucción por instrucción.

PROCESADORES DE LENGUAJE

Además de un compilador, pueden requerirse otros programas más para la creación de un

programa destino ejecutable, como se muestra en la figura 1.5. Un programa fuente puede dividirse en módulos guardados en archivos separados. La tarea de recolectar el programa de ori-
gen se confía algunas veces a un programa separado, llamado preprocesador. El preprocesador

también puede expandir algunos fragmentos de código abreviados de uso frecuente, llamados macros, en instrucciones del lenguaje fuente.

Además de un compilador, pueden requerirse otros programas más para la creación de un

programa destino ejecutable, como se muestra en la figura 1.5. Un programa fuente puede di-
vidirse en módulos guardados en archivos separados. La tarea de recolectar el programa de ori-
gen se confía algunas veces a un programa separado, llamado preprocesador. El preprocesador

también puede expandir algunos fragmentos de código abreviados de uso frecuente, llamados
macros, en instrucciones del lenguaje fuente.

LA ESTRUCTURA DE UN COMPILADOR

Hasta este punto, hemos tratado al compilador como una caja simple que mapea un programa

fuente a un programa destino con equivalencia semántica. Si abrimos esta caja un poco, podre-
mos ver que hay dos procesos en esta asignación: análisis y síntesis.

La parte del análisis divide el programa fuente en componentes e impone una estructura
gramatical sobre ellos

A menudo, los programas extensos se compilan en partes, por lo que tal vez haya que enlazar (vincular) el código máquina relocalizable con otros archivos objeto relocalizables y

archivos de biblioteca para producir el código que se ejecute en realidad en la máquina. El en-
lazador resuelve las direcciones de memoria externas, en donde el código en un archivo puede

hacer referencia a una ubicación en otro archivo. Entonces, el cargador reúne todos los archivos objeto ejecutables en la memoria para su ejecución.

Después utiliza esta estructura para crear una representación intermedia del programa fuente. Si la parte del análisis detecta que el programa fuente está mal formado en cuanto a la sintaxis, o que no tiene una semántica consistente, entonces debe proporcionar mensajes informativos para que el usuario pueda corregirlo. La parte del análisis también

recolecta información sobre el programa fuente y la almacena en una estructura de datos lla-
mada tabla de símbolos, la cual se pasa junto con la representación intermedia a la parte de la

síntesis.

PROCESADORES DE LENGUAJE

A la primera fase de un compilador se le llama análisis de léxico o escaneo. El analizador de léxico Lee el flujo de caracteres que componen el programa fuente y los agrupa en secuencias signifi-
cativas, conocidas como lexemas. Para cada lexema, el analizador léxico produce como salida
un token de la forma:

nombre-token, valor-atributo

que pasa a la fase siguiente, el análisis de la sintaxis. En el token, el primer componente nom-
bre-token es un símbolo abstracto que se utiliza durante el análisis sintáctico, y el segundo

componente valor-atributo apunta a una entrada en la tabla de símbolos para este token. La
información de la entrada en la tabla de símbolos se necesita para el análisis semántico y la
generación de código

La segunda fase del compilador es el análisis sintáctico o parsing. El parser (analizador sintác-
tico) utiliza los primeros componentes de los tokens producidos por el analizador de léxico para

crear una representación intermedia en forma de árbol que describa la estructura gramatical

del flujo de tokens. Una representación típica es el árbol sintáctico, en el cual cada nodo inte-
rior representa una operación y los hijos del nodo representan los argumentos de la operación.

El analizador semántico utiliza el árbol sintáctico y la información en la tabla de símbolos para

comprobar la consistencia semántica del programa fuente con la definición del lenguaje. Tam-
bién recopila información sobre el tipo y la guarda, ya sea en el árbol sintáctico o en la tabla

de símbolos, para usarla más tarde durante la generación de código intermedio.
Una parte importante del análisis semántico es la comprobación (verificación) de tipos, en
donde el compilador verifica que cada operador tenga operandos que coincidan. Por ejemplo,
muchas definiciones de lenguajes de programación requieren que el índice de un arreglo sea
entero; el compilador debe reportar un error si se utiliza un número de punto flotante para
indexar el arreglo.

En el proceso de traducir un programa fuente a código destino, un compilador puede construir
una o más representaciones intermedias, las cuales pueden tener una variedad de formas. Los

árboles sintácticos son una forma de representación intermedia; por lo general, se utilizan du-
rante el análisis sintáctico y semántico.

Después del análisis sintáctico y semántico del programa fuente, muchos compiladores ge-
neran un nivel bajo explícito, o una representación intermedia similar al código máquina, que

podemos considerar como un programa para una máquina abstracta. Esta representación in-
termedia debe tener dos propiedades importantes: debe ser fácil de producir y fácil de traducir

en la máquina destino.

La fase de optimización de código independiente de la máquina trata de mejorar el código in-
termedio, de manera que se produzca un mejor código destino. Por lo general, mejor significa

más rápido, pero pueden lograrse otros objetivos, como un código más corto, o un código de

destino que consuma menos poder. Por ejemplo, un algoritmo directo genera el código inter-
medio (1.3), usando una instrucción para cada operador en la representación tipo árbol que

produce el analizador semántico.

PROCESADORES DE LENGUAJE

El generador de código recibe como entrada una representación intermedia del programa fuente y la asigna al lenguaje destino. Si el lenguaje destino es código máquina, se seleccionan

registros o ubicaciones (localidades) de memoria para cada una de las variables que utiliza el programa. Después, las instrucciones intermedias se traducen en secuencias de instrucciones de máquina que realizan la misma tarea. Un aspecto crucial de la generación de código es la asignación juiciosa de los registros para guardar las variables.

El primer paso hacia los lenguajes de programación más amigables para las personas fue el

desarrollo de los lenguajes ensambladores a inicios de la década de 1950, los cuales usaban mnemónicos. Al principio, las instrucciones en un lenguaje ensamblador eran sólo representaciones

mnemónicas de las instrucciones de máquina. Más adelante, se agregaron macro instrucciones a los lenguajes ensambladores, para que un programador pudiera definir abreviaciones parametrizadas para las secuencias de uso frecuente de las instrucciones de máquina.

Una función esencial de un compilador es registrar los nombres de las variables que se utilizan en el programa fuente, y recolectar información sobre varios atributos de cada nombre. Estos atributos pueden proporcionar información acerca del espacio de almacenamiento que se asigna para un nombre, su tipo, su alcance (en qué parte del programa puede usarse su valor), y en

el caso de los nombres de procedimientos, cosas como el número y los tipos de sus argumentos, el método para pasar cada argumento (por ejemplo, por valor o por referencia) y el tipo

devuelto.

El tema sobre las fases tiene que ver con la organización lógica de un compilador. En una implementación, las actividades de varias fases pueden agruparse en una pasada, la cual lee un

archivo de entrada y escribe en un archivo de salida. Por ejemplo, las fases correspondientes al front-end del análisis léxico, análisis sintáctico, análisis semántico y generación de código intermedio podrían agruparse en una sola pasada. La optimización de código podría ser una pasada opcional. Entonces podría haber una pasada de back-end, consistente en la generación de código para una máquina de destino específica.

Al igual que cualquier desarrollador de software, el desarrollador de compiladores puede utilizar para su beneficio los entornos de desarrollo de software modernos que contienen herramientas como editores de lenguaje, depuradores, administradores de versiones, profilers, ambientes

eguros de prueba, etcétera. Además de estas herramientas generales para el desarrollo de software, se han creado otras herramientas más especializadas para ayudar a implementar las diversas fases de un compilador.

Las primeras computadoras electrónicas aparecieron en la década de 1940 y se programaban en lenguaje máquina, mediante secuencias de 0's y 1's que indicaban de manera explícita a la computadora las operaciones que debía ejecutar, y en qué orden. Las operaciones en sí eran de muy bajo nivel: mover datos de una ubicación a otra, sumar el contenido de dos registros, comparar dos valores, etcétera. Está demás decir, que este tipo de programación era lenta, tediosa y propensa a errores. Y una vez escritos, los programas eran difíciles de comprender y modificar.