Traductores de Lenguaje

Procesadores de lenguaje

Compilador

Es un programa que puede leer un programa en un lenguaje (el lenguaje fuente) y traducirlo en un programa equivalente en otro lenguaje (el lenguaje destino).

Una función importante del compilador es reportar cualquier error en el programa fuente que detecte durante el proceso de traducción.

Si el programa destino es un programa ejecutable en lenguaje máquina, entonces el usuario puede ejecutarlo para procesar las entradas y producir salidas (resultados).

Intérprete

Es otro tipo común de procesador de lenguaje.

En vez de producir un programa 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.

Por lo regular, el intérprete puede ofrecer mejores diagnósticos de error que un compilador, ya que ejecuta el programa fuente instrucción por instrucción.

La estructura de un compilador

Si abrimos esta caja un poco, podremos ver que hay dos procesos en esta asignación.

Análisis

Divide el programa fuente en componentes e impone una estructura gramatical sobre ellas.

Síntesis

Construye el programa destino deseado a partir de la representación intermedia y de la información en la tabla de símbolos.

Si examinamos el proceso de compilación con más detalle, podremos ver que opera como una secuencia de fases, cada una de las cuales transforma una representación del programa fuente en otro.

Análisis de léxico

Es la primera fase de un compilador.

El analizador de léxico lee el flujo de caracteres que componen el programa fuente y los agrupa en secuencias significativas, conocidas como lexemas. Para cada lexema, el analizador léxico produce como salida un token.

El analizador léxico ignora los espacios en blanco que separan a los lexemas.

Análisis sintáctico

La segunda fase del compilador.

El parser (analizador sintáctico) 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 interior representa una operación y los hijos del nodo representan los argumentos de la operación.

Análisis semántico

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.

También 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.

Generación de código intermedio

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.

Después del análisis sintáctico y semántico del programa fuente, muchos compiladores generan un nivel bajo explícito, o una representación intermedia similar al código máquina, que podemos considerar como un programa para una máquina abstracta.

Optimización de código

La fase de optimización de código independiente de la máquina trata de mejorar el código intermedio, de manera que se produzca un mejor código destino.

Un algoritmo simple de generación de código intermedio, seguido de la optimización de código, es una manera razonable de obtener un buen código de destino.

Generación de código

El generador de código recibe como entrada una representación intermedia del programa fuente y la asigna al lenguaje destino.

Un aspecto crucial de la generación de código es la asignación juiciosa de los registros para guardar las variables.

Administración de la tabla de símbolos

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.

La tabla de símbolos es una estructura de datos que contiene un registro para cada nombre de variable, con campos para los atributos del nombre.

La estructura de datos debe diseñarse de tal forma que permita al compilador buscar el registro para cada nombre, y almacenar u obtener datos de ese registro con rapidez.

El agrupamiento de fases en pasadas

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.

Algunas colecciones de compiladores se han creado en base a representaciones intermedias diseñadas con cuidado, las cuales permiten que el front-end para un lenguaje específico se interconecte con el back-end para cierta máquina destino.

· Herramientas de construcción de compiladores

Éstas herramientas utilizan lenguajes especializados para especificar e implementar componentes específicos, y muchas utilizan algoritmos bastante sofisticados.

Las herramientas más exitosas son las que ocultan los detalles del algoritmo de generación y producen componentes que pueden integrarse con facilidad al resto del compilador.

Algunas herramientas de construcción de compiladores de uso común son.

- · Generadores de analizadores sintácticos (parsers).
- · Generadores de escáneres.
- · Motores de traducción orientados a la sintaxis.
- · Generadores de generadores de código.
- · Motores de análisis de flujos de datos.
- · Kits (conjuntos) de herramientas para la construcción de compiladores.

La evolución de los lenguajes de programación

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.

El avance a los lenguajes de alto nivel

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.

Un paso importante hacia los lenguajes de alto nivel se hizo en la segunda mitad de la década de 1950.

En las siguientes décadas se crearon muchos lenguajes más con características innovadoras para facilitar que la programación fuera más natural y robusta.

En la actualidad existen miles de lenguajes de programación. Pueden clasificarse en una variedad de formas.

Impactos en el compilador

Desde su diseño, los lenguajes de programación y los compiladores están íntimamente relacionados; los avances en los lenguajes de programación impusieron nuevas demandas sobre los escritores de compiladores.

Los compiladores pueden ayudar a promover el uso de lenguajes de alto nivel, al minimizar la sobrecarga de ejecución de los programas escritos en estos lenguajes.

Un compilador debe traducir en forma correcta el conjunto potencialmente infinito de programas que podrían escribirse en el lenguaje fuente.

SERGIO GARCÍA LEÓN