Instituto de Ciencias Básicas e Ingeniería

Profesor: Dr. Eduardo Cornejo Velázquez
Materia: Autómatas y Compiladores
Alumno: Roberto Martínez Magaña
Licenciatura en Ciencias Computacionales
Semestre: enero - junio 2025
6°3

CAPÍTULO 1A

PROCESADORES DE LEMGUAJE

pueden

La tarea de recolectar el programa de ori gen se confía algunas veces a un programa separado, llamado preprocesador.

incluso

El lenguaje ensamblador se procesa mediante un programa llamado ensamblador, el cual produce código máquina relocalizable como su salida.

al igual

El enlazador resuelve las direcciones de memoria externas, en donde el código en un archivo puede hacer referencia a una ubicación en otro archivo.

encima

el cargador reúne todos los archivos objeto ejecutables en la memoria para su ejecución.

explica que

Un 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);

además de

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

también

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

asimismo

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,

ESTRUCTURA DE UN COMPILADOR

se entiende como

Al compilador como una caja simple que mapea un programa fuente a un programa destino con equivalencia semántica.

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

La parte de la síntesis construye el programa destino deseado a partir de la representación intermedia y de la información en la tabla de símbolos.

análisis (front-end del compilador)
síntesis (traducción back.-end)

además

Algunos compiladores tienen una fase de optimización de código independiente de la máquina, entre el front-end y el back-end.

de igual modo

El propósito de esta optimización es realizar transformaciones sobre a representación intermedia, para que el back-end pueda producir un mejor programa destino de lo que hubiera producido con una representación intermedia sin optimizar.

ANÁLISIS DE LÉXICO

se entiende como

La primera fase de un compilador se le llama análisis de léxico o escaneo.

se compone de

lee el flujo de caracteres que componen el programa fuente y los agrupa en secuencias significativas conocidas como lexemas.

provoca

{nombre-token, valor-atributo}

además

Análisis de la sintaxis. En el token, el primer componente nombre-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.

ANÁLISIS SINTÁCTICO

se entiende como

La segunda fase del compilador es el análisis sintáctico o parsing.

trabaja de manera que

El parser (analizador sintáctico) utiliza los primeros componentes de los tokens producidos por el analizador de léxico paracrear una representación intermedia en forma de árbol que describa la estructura gramatical del flujo de tokens.

también

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

ANÁLISIS SINTÁCTICO

se entiende como

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

Una parte importante del análisis semántico es la comprobación (verificación) de tipos, en donde el compilador verifica que cada operador tenga operandos que coincidan.

aún

La especificación del lenguaje puede permitir ciertas conversiones de tipo conocidas como coerciones.

GENERACIÓN DE CÓDIGO INTERMEDIO

se entiende como

En el proceso de traducción de un programa, un compilador puede generar representaciones intermedias, como los árboles sintácticos, que se usan durante los análisis sintáctico y semántico.

también

el código de tres direcciones, que consiste en una secuencia de instrucciones similares a ensamblador, con tres operandos por instrucción, y cada operando puede actuar como un registro.

- Un operador del lado derecho.
- Generar un nombre temporal temporal para guardar el valor calculado por una instrucción de 3 direcciones.
- "instrucciones de 3 direcciones como la primera y la última en la secuencia"

OPTIMIZACIÓN DE CÓDIGO

se entiende como

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 optimización es efectivo para generar buen código de destino.

Los compiladores optimizadores dedican mucho tiempo a esta fase, realizando optimizaciones que mejoran considerablemente el tiempo de ejecución sin afectar demasiado la velocidad de la compilación.

OPTIMIZACIÓN DE CÓDIGO

se entiende como

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.

TABLA DE SÍMBOLOS Y FASES EN PASADAS

se define

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

también

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 unarchivo 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.

front-end (para un lenguaje específico) back-end (para una máquina destino).

HERRAMIENTAS DE CONSTRUCCIÓN DE COMPILADORES

se define

los desarrolladores de compiladores pueden aprovechar herramientas modernas como editores de lenguaje, depuradores, administradores de versiones, profilers y entornos de prueba seguros.

tiene

- **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**

EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

explica

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.

además

- **Primera generación:** Lenguajes de máquina.
- **Segunda generación:** Lenguajes ensambladores.
- **Tercera generación:** Lenguajes de alto nivel como Fortran, Cobol, Lisp, C, C++, C#, Java.
- **Cuarta generación:** Lenguajes especializados para aplicaciones, como SQL y Postscript.
- **Quinta generación:** Lenguajes basados en lógica, como Prolog.

se clasifican

Los lenguajes también se pueden clasificar como imperativos (como C, C++, C#, Java), que definen cómo se realiza un cálculo, o declarativos (como Prolog), que especifican qué cálculo se debe realizar.

Los lenguajes de secuencias de comandos (como Awk, JavaScript, Perl, PHP, Python, Ruby y Tcl) son lenguajes interpretados y diseñados para unir cálculos, siendo generalmente más cortos que los escritos en lenguajes como C.

Los lenguajes orientados a objetos (como Simula 67, Smalltalk, C++, C#, Java y Ruby) permiten programar mediante objetos que interactúan entre sí.

IMPACTOS EN EL COMPILADOR

explica

los lenguajes de programación y los compiladores han estado estrechamente relacionados. A medida que los lenguajes de programación evolucionaron, los compiladores tuvieron que adaptarse, desarrollando nuevos algoritmos y representaciones para soportar las características avanzadas de estos lenguajes y aprovechar al máximo las nuevas arquitecturas de hardware.

también

Los compiladores permiten el uso eficiente de lenguajes de alto nivel al minimizar la sobrecarga de ejecución, y son cruciales para aprovechar las arquitecturas computacionales de alto rendimiento.

Un compilador debe ser capaz de traducir correctamente cualquier programa escrito en el lenguaje fuente, aunque generar el código destino óptimo es un problema indecidible. Los compiladores deben hacer concesiones y utilizar heurísticas para generar código eficiente.