COMPILADORES
Unidad I: Introducción al proceso de compilación

Flor Prof. Flor Narciso
GIDyC-Departamento de Computación
LABSIULA-Escuela de Ingeniería de Sistemas
Facultad de Ingeniería
Universidad de Los Andes
fnarciso@ula.ve

Lenguaje de programación

Un ***lenguaje de programación*** se puede definir de diferentes maneras:

- Notación formal para describir algoritmos y funciones que serán ejecutados por una computadora.
- Lenguaje para comunicar instrucciones a una computadora.
- Convención para escribir descripciones que puedan ser evaluadas.

Clasificación de los lenguajes de programación (según el grado de independencia de la máquina)

- **Lenguaje de máquina**
  - La forma más baja de un lenguaje de programación.
  - La notación que entiende directamente una computadora.
  - Binario o hexadecimal.
  - Cada instrucción se representa:
    - Un código numérico y
    - Unas direcciones de memoria.
  - Arquitectura de la máquina de Von Neumann. Conjunto de instrucciones basado en:
    - Datos
    - Operaciones aritméticas y lógicas
    - Asignaciones de posiciones de memoria
    - Control de flujo

Clasificación de los lenguajes de programación (según el grado de independencia de la máquina)

- **Lenguaje ensamblador**
  - Versión simbólica de un lenguaje de máquina:
    - Cada código de operación se indica por un código simbólico: ADD, MUL...
    - Asignaciones de memoria se dan con nombres simbólicos

Clasificación de los lenguajes de programación (según el grado de independencia de la máquina)

- **Lenguaje de nivel intermedio**
  - Características de los lenguajes máquina:
    - Acceso directo a posiciones memoria
    - Almacenar variables en registros del procesador
  - Características de lenguajes de alto nivel:
    - Manejo de estructuras de control
    - Manejo de datos

Ejemplo: Lenguaje C

Clasificación de los lenguajes de programación (según el grado de independencia de la máquina)

- **Lenguaje de alto nivel**
  - Características superiores a los lenguajes ensambladores
  - No acceso directo al sistema
  - Estructura de datos complejas
  - Utilización de bloques, procedimientos o subrutinas

Ejemplos: Ada, ALGOL, Basic, C, C++, C#, Clipper, Cobol, Fortran, Java, Lexico, Logo, Object Pascal, Pascal, Perl, PHP, PL/SQL, Phyton, Modula 2.

Lenguajes funcionales: Haskell, Lisp **TAREA!!!!**

Clasificación de los lenguajes de programación (según el grado de independencia de la máquina)

- **Lenguaje orientado a problemas concretos**
  - Resolución de problemas en un campo específico.

Ejemplos: SQL, XBASE, Postscript

Procesar

- Someter a un *proceso de transformación* mediante operaciones programadas. Es decir, transformar un origen (*fuente*) en un destino mediante una herramienta de transformación que permita operaciones programadas: *la computadora*.

Procesador de lenguaje

- Nombre *genérico* que reciben las *aplicaciones informáticas* en las que uno de los datos fundamentales de *entrada es un lenguaje*:
  - Traductores
  - Ensambladores
  - Cargadores
  - Desensambladores
  - Depuradores
  - Optimizadores de código
  - Preprocesadores
  - Editores
  - Compiladores
  - Enlazadores
  - Intérpretes
  - Decompiladores
  - Analizadores de rendimiento
  - Compresores
  - Formateadores

TAREA!!!!

Procesador de lenguaje

Se tomará como paradigma de los procesadores de lenguaje los ***compiladores***.

Los lenguajes de alto nivel hicieron necesarios los compiladores a partir de los años 50. Desde entonces, gracias al descubrimiento de técnicas sistemáticas para el manejo de muchas tareas que surgen en la compilación, al desarrollo de buenos lenguajes de implantación, entornos de programación y herramientas de software, el diseño y desarrollo de un compilador se ha simplificado enormemente.

Traductor

- Lee un texto fuente y lo traduce a un texto objeto.
- Está escrito en un lenguaje de Implantación (LI). Puede ser cualquier lenguaje desde uno de alto nivel a uno máquina.
- El texto fuente está escrito en lenguaje fuente (LF). Normalmente uno de alto nivel pero también puede ser de bajo nivel.
- El texto objeto está escrito en lenguaje objeto (LO). Puede ser otro lenguaje de alto nivel, un lenguaje máquina o un ensamblador.

Compilador

Traductor que acepta programas escritos en un **lenguaje de programación de alto nivel y los traduce a otro lenguaje**, generando un programa equivalente independiente, que puede ejecutarse tantas veces como se desee.

Compilación

- Proceso por el cual se traducen programas en código fuente (programa fuente) a programas en código objeto (programa objeto).
- El programa que realiza esta traducción se llama compilador.
- El archivo de código objeto que se obtiene con la compilación está representado normalmente en código de máquina, aunque también puede ser un código intermedio binario multiplataforma (*bytecode*).

Compilación

- El tiempo que se tarda en traducir un programa en código fuente se llama **tiempo de compilación.**
- El tiempo que tarda en ejecutarse un programa en código objeto se llama **tiempo de ejecución**.
- El programa fuente y los datos se procesan en momentos diferentes.

Código fuente

- Conjunto de líneas de código que conforman un bloque de texto que normalmente genera otro código mediante un compilador o intérprete para ser ejecutado por una computadora.
- Normalmente se refiere a la programación de software. Un único programador o un equipo de programadores escriben el código fuente en el lenguaje de programación elegido. Posteriormente en un proceso de compilación el código fuente se traduce en código objeto.

Código objeto

Código resultante de la compilación del código fuente, por lo general está codificado en código de máquina y distribuido en varios archivos resultantes de la compilación de cada archivo de código fuente.

Compiladores

Los compiladores son las herramientas más utilizadas por los informáticos para el desarrollo de aplicaciones.

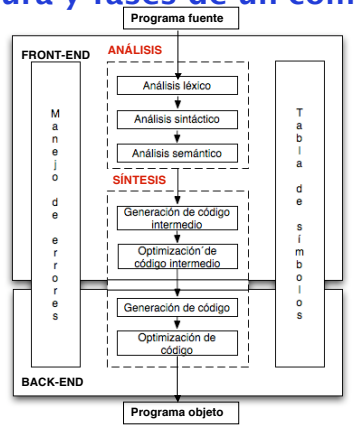
En el caso particular del desarrollo de compiladores se hace necesario definir tres aspectos básicos:

- El léxico, la sintaxis y la semántica del lenguaje fuente a ser compilado.
- La estructura interna del compilador.
- La arquitectura de la computadora y su conjunto de instrucciones del lenguaje objeto.

Estructura y fases de un compilador

- La construcción de un compilador para un determinado lenguaje es una tarea compleja, que se puede reducir siguiendo una *metodología*: dividir en módulos las diferentes fases.
- La complejidad dependerá de las características del lenguaje fuente y del lenguaje objeto y de su diferencia de niveles.

Estructura y fases de un compilador



Estructura y fases de un compilador

Fases de un compilador:

Análisis : Comprobar la corrección del programa fuente
- Léxico
- Sintáctico
- Semántico

Síntesis
- Generación de código intermedio
- Optimización de código intermedio
- Generación de código
- Optimización de código

Estructura y fases de un compilador

Fases de un compilador:

Análisis léxico: Se realiza en el nivel de los caracteres, debe reconocer tokens y entregarlos junto con sus atributos al analizador sintáctico, aunque estos últimos no son necesarios para el análisis sintáctico, sino para las fases siguientes.

El programa fuente es tratado con el analizador léxico (scanner), lee secuencialmente caracteres, los compara con patrones que representan unidades sintácticas e identifica éstas, también llamadas componentes léxicos o tokens, tales como: constantes, identificadores (de variables, de funciones, de procedimientos, de tipos, etc.), palabras reservadas y operadores. Una vez identificado el token es entregado al analizador sintáctico. A cada token se le asocia una serie de informaciones, según las necesidades del traductor.

Estructura y fases de un compilador

Fases de un compilador:

Análisis sintáctico: Llamado también parser, realiza su análisis en el nivel de la sentencia.

Es mucho más complejo que el análisis léxico.

Su función es tomar los tokens que ha encontrado el analizador léxico y determinar la estructura sintáctica de las sentencias, agrupando los tokens en clases sintácticas (los no terminales de la gramática), tales como expresiones, funciones, etc.

Estructura y fases de un compilador

Fases de un compilador:

Análisis semántico: Detecta la validez semántica (reglas de significado) de las sentencias aceptadas por el sintáctico. Típico de esta fase es la comprobación de tipos de datos.

Estructura y fases de un compilador

Fases de un compilador:

Generación de código intermedio: El código intermedio no es un lenguaje de programación de ninguna máquina real, sino que corresponde a una máquina abstracta, que se debe definir lo más general posible, de manera que sea posible traducir este código intermedio a cualquier máquina real.

El objetivo del código intermedio es reducir el número de programas necesarios para construir traductores y permitir más fácilmente la transportabilidad de los traductores desde unas máquinas a otras.

Estructura y fases de un compilador

Fases de un compilador:

Optimización de código intermedio: Es independiente de la máquina. Algunas optimizaciones pueden consistir en evaluación de expresiones constantes, el uso de las propiedades asociativa, conmutativa de algunos operadores, reducción de expresiones comunes, etc.

Generación de código: Una vez que se ha obtenido el código intermedio se pasará a ensamblador o a código máquina de una máquina real en el caso de un compilador o a otro lenguaje en el caso de un traductor.

Estructura y fases de un compilador

Fases de un compilador:

Optimización de código: En este caso ya depende de la máquina, de su arquitectura, de la asignación óptima de registros, el uso de operaciones de registros en vez de usar memoria.

Manejo de errores: Los errores encontrados en la diferentes fase de análisis se envían a un módulo de manejo de errores. En el caso más sencillo, sacará un mensaje indicando el error, el número de línea donde se ha producido y abortará el proceso de análisis o traducción.

Puede sofisticarse este módulo si se intenta recuperar el error (una especie de reparación provisional) para continuar el proceso el mayor tiempo posible y encontrar el máximo de errores.

Estructura y fases de un compilador

Fases de un compilador:

Tabla de símbolos: Es una estructura de datos que contiene toda la información relativa a cada identificador que aparece en el programa fuente. Cada elemento de la tabla se compone de al menos del identificador y sus atributos.

Los atributos son informaciones relativas a cada identificador, necesarias para o bien realizar el análisis semántico o bien la traducción. Cualquiera de los tres analizadores puede rellenar algún atributo, pero el nombre del identificador (lexema) es misión del analizador léxico.

Estructura y fases de un compilador

- Estructura de un compilador:
  - ***FRONT-END***: Analiza el código fuente, comprueba su validez, genera el árbol de derivación y rellena los valores de la tabla de símbolos. Suele ser independiente de la plataforma o sistema para el cual se vaya a compilar.
  - ***BACK-END***: Genera el código de máquina, específico de una plataforma, a partir de los resultados de la fase de análisis, realizada por el *FRONT-END*.

Estructura y fases de un compilador

- Estructura de un compilador:
  - Esta división permite que el mismo *BACK-END* se utilice para generar el código de máquina de varios lenguajes de programación distintos y que el mismo *FRONT-END* que sirve para analizar el código fuente de un lenguaje de programación concreto sirva para la generación de código de máquina en varias plataformas distintas.
  - El código que genera el *BACK END* normalmente no se puede ejecutar directamente, sino que necesita ser enlazado por un programa enlazador (*linker*).

Tipos de compiladores

- Esta taxonomía de los tipos de compiladores no es excluyente, por lo que puede haber compiladores que se adscriban a varias categorías:
  - **Compiladores cruzados**: Se ejecutan en una máquina pero el código objeto que producen es para otra máquina.
  - **Compiladores optimizadores**: Realizan cambios en el código para mejorar su eficiencia, pero manteniendo la funcionalidad del programa original.
  - **Compiladores de una sola pasada**: Generan el código máquina a partir de una única lectura del código fuente.

Tipos de compiladores

- **Compiladores de varias pasadas**: Necesitan leer el código fuente varias veces antes de poder producir el código de máquina.
- **Compiladores JIT** (*Just In Time*): Forman parte de un intérprete y compilan partes del código según se necesitan.

Tarea: Investigar acerca de los compiladores de varias pasadas

Herramientas para la construcción de procesadores de lenguaje

- **Generadores de analizadores léxicos**: Basada en el uso de expresiones regulares, generan automáticamente el código fuente para el análisis léxico a partir de una especificación de los tokens. La más usada es **lex**, incorporada en el sistema operativo UNIX. Existen versiones para PC.
- **Generadores de analizadores sintácticos**: Construyen el código fuente del analizador sintáctico a partir de la especificación de la gramática del lenguaje fuente. La mas usada **yacc** incluida en UNIX. Tambien existen versiones para PC.

Herramientas para la construcción de procesadores de lenguaje

- **Analizadores de gramáticas**: Dada una gramática especificada formalmente, verifican si es de un determinado tipo o no. Normalmente se utilizan para verificar las gramáticas LL(k) y LR(k).
- **Máquinas de traducción dirigida por sintaxis**: Producen un conjunto de rutinas que recorren el árbol sintáctico y generan código intermedio. Asocian una o más traducciones a cada nodo sintáctico.

Herramientas para la construcción de procesadores de lenguaje

- **Generadores automáticos de código**: Trabajan con un conjunto de reglas que permiten la traducción del código en lenguaje intermedio al lenguaje objeto. Las reglas suelen remplazar instrucciones de código intermedio por patrones que contienen las instrucciones equivalentes de la máquina objeto.
- **Analizadores de flujo**: Suministran la información necesaria para realizar las optimizaciones de código.

Aplicaciones de los procesadores de lenguaje

- Las técnicas empleadas en la construcción de traductores, compiladores e intérpretes pueden aplicarse en la construcción de otras herramientas:
  - **Editores sensibles al contexto**
  - **Conversores de formato**
  - **Preprocesadores**
  - **Formateadores de código fuente**
  - **Generadores de código**
  - **Verificación estática de programas**

Aplicaciones de los procesadores de lenguaje

- **Formateadores de texto**
- **Intérpretes de comandos de un sistema operativo**
- **Construcción de entornos operativos**
- **Intérpretes para consultar base de datos**
- **Compiladores de silicio**
- **Procesamiento de lenguajes naturales**
- **Reconocimiento del habla**