TEMA-3.pdf

Tema3_CompilaciónyEnlazadodeprogramas

1° Fundamentos del Software

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

TEMA 3: COMPILACIÓN Y ENLAZADO DE PROGRAMAS

1. Lenguajes de programación

Un lenguaje de programación es un conjunto de símbolos y de reglas para combinarlos, que se usan para expresar algoritmos. Estos son independientes de la arquitectura física del computador, lo que favorece la portabilidad de los programas. Un lenguaje de alto nivel utiliza notaciones fácilmente reconocible por las personas en el ámbito en que se usan. Una instrucción en un lenguaje de alto nivel da lugar, tras el proceso de traducción, a varias instrucciones en lenguaje máquina (sistema de códigos directamente interpretable por un circuito microprogramable).

2. Construccion de traductores

2.1 Definición de gramática

Un conjunto finito de reglas (la gramática del lenguaje), para la construcción de las sentencias correctas del lenguaje. Una gramática proporciona una especificación sintáctica precisa de un lenguaje de programación. La complejidad de la verificación sintáctica depende del tipo de gramática que define el lenguaje. Una gramática se define como $G = (V_N, V_T, P, S)$ donde:

- V_N es el conjunto de símbolos no terminales.
- V_T es el conjunto de símbolos terminales.
- P es el conjunto de producciones o reglas gramaticales.
- S es el símbolo inicial (es un símbolo no terminal) .

2.2 Definición de traductor

Un traductor es un programa que recibe como entrada un texto en un lenguaje de programación concreto y produce, como salida, un texto en lenguaje máquina equivalente. Existen dos tipos de traductores: Compiladores e intérpretes.

No nos vemos pero ahora estamos más cerca #QuédateEnCasa

Primera academia especializada en estudios de la Facultad de Informática UCM

academia@mathsinformática.com

C/Andrés Mellado, 88 duplicado

www.mathsinformatica.com

academia.maths

91 399 45 49

615 29 80 22

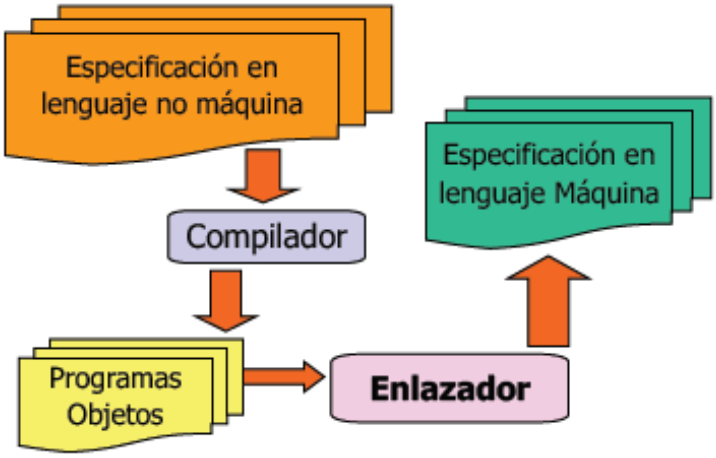
Material online

- Resolución de ejercicios en vídeo
- Clases virtuales Skype y Hangouts
- Grupos de asignatura en Whastssap

Maths informática

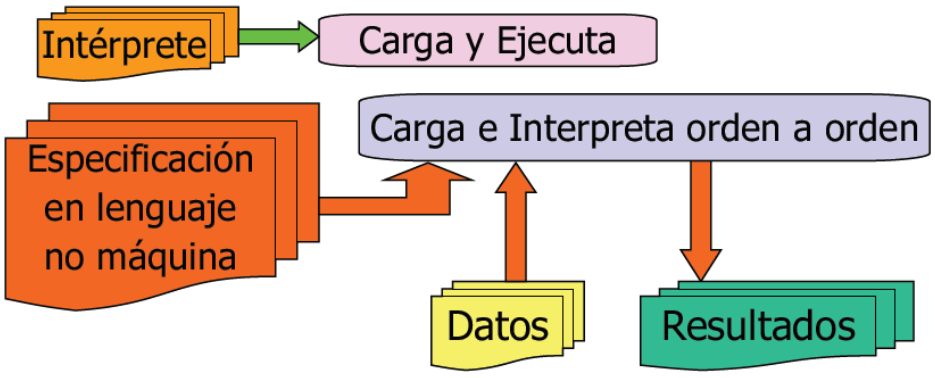
2.2.1 Compilador

Es un Software que traduce un programa escrito en un lenguaje de programación de alto nivel (C++, C, etc.) en lenguaje de máquina. Un compilador generalmente genera lenguaje ensamblador primero y luego traduce el lenguaje ensamblador al lenguaje máquina. Una utilidad conocida como «enlazador» combina todos los módulos de lenguaje de máquina necesarios en un programa ejecutable que se puede ejecutar en la computadora.



2.2.2 Intérprete

Un intérprete es un programa que ejecuta línea a línea las instrucciones de un programa de alto nivel. El intérprete carga el código fuente y traduce las instrucciones a un lenguaje intermedio que puede luego ser ejecutado. Los intérpretes generan un código binario que se interpreta cada vez que se ejecuta el programa a diferencia del compilador que crea un archivo ejecutable. Los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa mientras se ejecuta, pero a cambio son más flexibles como entornos de programación y depuración, y permiten ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete, sino del propio intérprete (lo que se conoce comúnmente como máquina virtual).



2.3 Definición de gramática: conceptos previos

Alfabeto: Conjunto finito de símbolos.

Cadena: secuencia finita de símbolos de un alfabeto determinado.

Símbolos terminales: elementos de un alfabeto usados para crear cadenas.

Símbolos no terminales: variables sintáticas que representan conjuntos de cadenas. Se utilizan en las reglas gramaticales y no son elementos del alfabeto.

3. Proceso de compilación. Fases de traducción

Cualquier compilador debe realizar dos tareas principales: análisis del programa a compilar y síntesis de un programa en lenguaje maquina que, cuando se ejecute, realizara correctamente las actividades descritas en el programa fuente. Para el estudio de un compilador, es necesario dividir su trabajo en fases. Cada fase representa una transformación al código fuente para obtener el código objeto. La siguiente figura representa los componentes en que se divide un compilador. Las tres primeras fases realizan la tarea de análisis, y las demás la síntesis. En cada una de las fases se utiliza un administrador de la tabla de símbolos y un manejador de errores.

3.1 Análisis léxico

Un analizador léxico es la primera fase de un compilador, consistente en un programa que recibe como entrada el código fuente de otro programa y produce una salida compuesta de tokens o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico. Un token es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.

Un error léxico se produce cuando el analizador intenta reconocer componente léxicos y la cadena de caracteres de la entrada no encaja con ningun patrón. Son situaciones en las que se usa un caracter no válido (@, %,?...) que no pertenece al vocabulario del lenguaje de programación, al escribir mal un identificador o palabra reservada.

3.2 Análisis sintáctico

La tarea del analizador es, en este caso, la descomposición y transformación de las entradas en un formato utilizable para su posterior procesamiento. Se analiza una cadena de instrucciones en un lenguaje de programación y luego se descompone en sus componentes individuales.
Un error sintáctico se produce cuando se escribe código de una forma no admitida por las reglas del lenguaje. Los errores de sintaxis son detectados casi siempre por el compilador o intérprete, que muestra un mensaje de error que informa del problema.

3.3 Análisis semántico

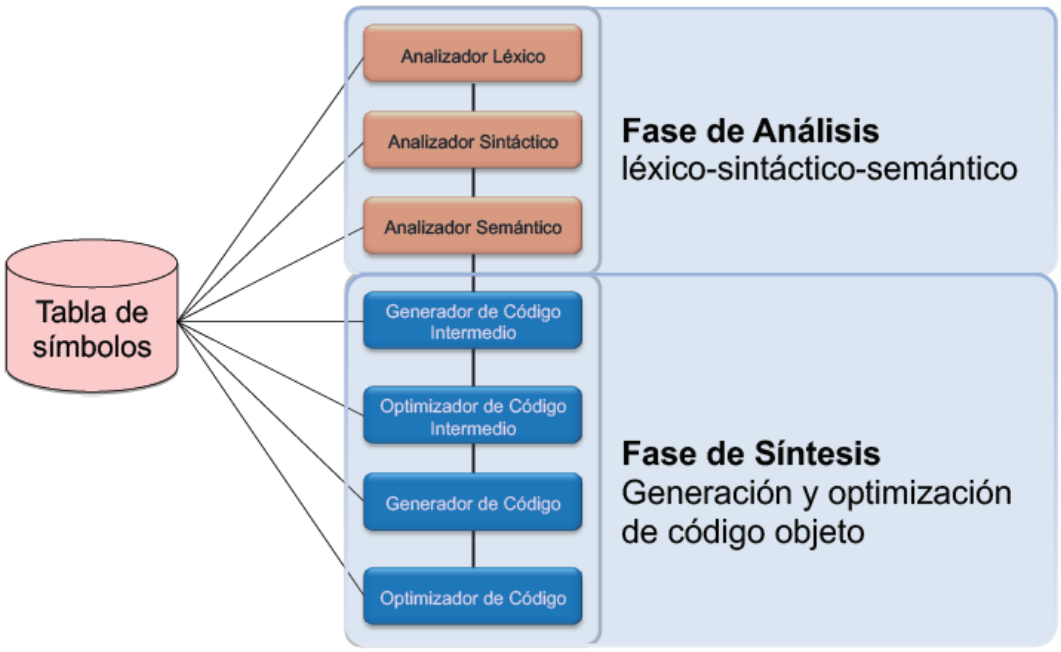
La semántica de un lenguaje de programación es el significado dado a las distintas construcciones sintácticas. En los lenguajes de programación, el significado está ligado a la estructura sintáctica de las sentencias.
Los errores semánticos son más sutiles. Un error semántico se produce cuando la sintaxis del código es correcta, pero la semántica o significado no es el que se pretendía. La construcción obedece las reglas del lenguaje, y por ello el compilador o intérprete no detectan los errores semánticos. Los compiladores e intérpretes sólo se ocupan de la estructura del código que se escribe, y no de su significado. Un error semántico puede hacer que el programa termine de forma anormal, con o sin un mensaje de error.

3.4 Generación de código

En esta fase se genera un archivo con un código en lenguaje objeto (generalmente lenguaje máquina) con el mismo significado que el texto fuente. En algunos se intercala una fase de generaciñon de código intermedio para proporcionar independencias de las fases de análisis con respecto al lenguaje máquina (portabilidad del compilador) o para hacer más fácil la optimización de código.

3.5 Optimización de código

La optimización de código es el conjunto de fases de un compilador que transforman un fragmento de código en otro fragmento con un comportamiento equivalente y que se ejecuta de forma más eficiente, es decir, usando menos recursos de cálculo como memoria o tiempo de ejecución.



4. Intérpretes

Un intérprete es un programa informático capaz de analizar y ejecutar otros programas. Los intérpretes se diferencian de los compiladores o de los ensambladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes solo realizan la traducción a medida que sea necesaria, tipicamente, instrucción por instrucción y normalmente no guardan el resultado de dicha traducción.

¿Cuando es útil un intérprete y cuando no?

SI es útil cuando:

- El programador trabaja en un entorno interactivo y se desean obtener los resultados de la ejecucion de una instruccion antes de ejecutar la siguiente.

- El programador lo ejecuta pocas veces y el tiempo de ejecución no importa.

- Las instrucciones del lenguaje tiene una estructura simple y pueden ser analizadas facilmente.

- Cada instruccion será ejecutada una sola vez.

NO es útil cuando:

- Las instrucciones del lenguaje son complejas.

- Los programas van a trabajar en modo de producción y la velocidad es importante.

- Las instrucciones serán ejecutadas con frecuencia.

5. Modelo de memoria de un proceso

Los elementos responsables de la gestión de memoria son:

- Lenguaje de programación.

- Compilador.

- Enlazador.

- Sistema operativo.

- Hardware para la gestión de memoria: MMU – Unidad de gestiñón de memoria.

5.1 Niveles de la gestión de memoria

Aquí existen tres niveles que de menos a mayor nivel de detalle serían:

- Nivel de proceso: Es responsabilidad del SO y se encarga del reparto de memoria entre los procesos que la sollicitan.

- Nivel de regiones: En este nivel de distribuye el espacio asignado a las regiones de un proceso. Esta gestionado por el SO y la división en regiones la realiza el compilador.

- Nivel de zonas: Se encarga la gestión del lenguaje de programación con ayuda del SO. Aquí se reparte cada región entre las diferentes zonas (nivel estático, dinámico basado en pila o en heap) de la memoria.

5.2 Necesidades de memoria de un proceso

Las necesidad de la memoria de un proceso son:

- Tener un espacio lógico independiente y protegido del resto de procesos.

- Posibilidad de compartir memoria, para poder comunicarse e intercambiar datos entre dos procesos.

- Soporte para diferentes regiones

- Facilidad de depuración, para poder solucionar errores de programación con mayor facilidad y rapidez.

- Uso de un mapa amplio de memoria (estructura de datos que indica como esta distribuid la memoria).

- Uso de diferentes tipos de objetos de memoria.

- Persistencia de datos(representación residual de **datos** que han sido de alguna manera nominalmente borrados o eliminados).

- Desarrollo modular (dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable).

- Carga dinámica de módulos. Esto ayuda a que los archivos que contengan un gran número de módulos e instrumentos de software puedan abrirse más rápido.

6. Modelo de memoria de un proceso

Estudiaremos aspectos relacionados con la gestión del mapa de memoria de un proceso, desde la generación del ejecutable a su carga en memoria.

6.1 Tipos de datos (desde el punto de vistea de su implementación en memoria)

- Datos estáticos

Su tamaño y forma es constante durante la ejecución de un programa y por tanto se determinan en tiempo de compilación. Estos pueden tener o no un valor inicial, lo cuál se puede implementar con direccionamiento absoluto o direccionamiento relativo.

- Datos dinámicos asociados a la ejecución de una función

Su tamaño y forma es variable (o puede serlo) a lo largo de un programa, por lo que se crean y destruyen en tiempo de ejecución. Se almacenan en pila en un registro de activación. Se crean al activar una función y se destruyen al terminar la misma.

- Datos dinámicos controlados por el programa

HEAP (zona de memoria usada en tiempo de ejecución para albergar los datos no conocidos en tiempo de compilación).

6.2 Ciclo de vida de un programa

El ciclo de vida de un programa es una secuencia estructurada y bien definida de las etapas para desarrollar el producto software deseado. A partir del código fuente, un programa debe pasar por varias fases antes de poder ejecutarse:

- Preprocesado: hacer intelible el código fuente para el compilador, cambiando las directivas de preprocesamiento por valores para el compilador. Estas directivas siempre estan señaladas por caracteres especiales para que solo sean modificadas por el preprocesador.

- Compilación: supone que un programa escrito en un cierto lenguaje de programación es convertido en un programa desarrollado en un lenguaje diferente, que suele ser un lenguaje máquina. La compilación se desarrolla en dos grandes fases: primero se analiza el programa fuente y luego se sintetiza el programa objeto. También asigna direcciones a los símbolos estáticos y resuelve las referencias de forma absoluta o relativa (necesita reubicación). Estas referencias se resuelven usando direccionamiento relativo a pila para datos relacionados a la invocación de una función, o con direccionamiento indirecto para el heap.

- Ensamblado: consiste en transformar un archivo escrito en lenguaje ensamblador en un archivo objeto o archivo binario.

- Enlazado: toma los objetos generados en la compilación y enlaza el código objeto con sus bibliotecas para producir un fichero ejecutable. Según el tipo de enlazado, se define una especia de ámbito. El enlazado externo genera una visibilidad global, el interno una visibilidad de fichero. Cuando no hay enlazado, se genera una visibilidad del bloque.

- Carga y ejecición: La reubicación del proceso se realiza en la carga (reubicación estática) o en ejecución (reubicación dinámica) y es función del Sistema Operativo ayudado por un hardware específico (MMU – Unidad de Gestión de Memoria). Depende del tipo de gestión de memoria que se realice: paginación o segmentación.

6.3 Diferencias entre archivo objeto y archivo ejecutable

Los archivos objeto son archivos intermedios generados por el compilador antes de crear un ejecutable. Cuál es la diferencia entre el archivo de objeto y el archivo ejecutable. La diferencia principal entre el archivo objeto y el archivo ejecutable es que un archivo de objeto es un archivo generado después de compilar el código fuente, mientras que un archivo ejecutable es un archivo generado después de vincular un conjunto de archivos de objetos mediante un enlazador.

7. Bibliotecas

Una biblioteca es una colección de objetos normalmente relacionados entre sí. Estas favorecen la modularidad y reusabilidad de un código. Podemos clasificarlas según la forma de enlazarlas:

- Bibliotecas estáticas: fichero contenedor con varios archivos de código objeto empaquetados, que en el proceso de enlazado durante la compilación serán copiados y relocalizados (si es necesario) en el fichero ejecutable final, junto con el resto de ficheros de código objeto. Algunos de los inconvenientes de las bibliotecas estáticas son:

1. El codigo de la biblioteca está en todos los ejecutables que la usan, por lo que se desperdicia disco y memoria principal.

2. Al actualizar la biblioteca, hay que recompilar todos los programas que la usan para que se puedan beneficiar de la nueva versión.

3. Producen ejecutables de gran tamaño.

- Bibliotecas dinámicas: ficheros que contienen código objeto construido de forma independiente de su ubicación de tal modo que están preparadas para poder ser requeridas y cargadas en tiempo de ejecución por cualquier programa, en lugar de tener que ser enlazadas, previamente, en tiempo de compilación. El archivo correspondiente se diferencia de un archivo ejecutable en los siguientes aspectos:

1. Contiene información de reubicación.
2. Contiene una tabla de símbolos.
3. En la cabezera no se almacena información de punto de entrada.

8. Automaticación del proceso de compilacion y enlazado

Automatizar la construcción es la técnica utilizada durante el ciclo de vida de desarrollo de software donde la transformación del código fuente en el ejecutable se realiza mediante un guión (script).

La automatización mejora la calidad del resultado final y permite el control de versiones. Varias formas son con una herramiento make (makefile) o con un IDE (Entornos de desarrollo integrados) que embebe los guiones y el proceso de compilación y enlazado (CodeBlocks).