Fundamentos del Software

CONCEPTOS IMPORTANTES

Tema 3 - Compilación y enlazado de programas

La cuestión central de este tema es la creación de un archivo ejecutable. Siempre partiremos de un programa creado con lenguaje de programación que se traduce a lenguaje máquina para generar un ejecutable que el ordenador entienda.

Traductor => Programa que recibe como entrada un texto en lenguaje de programación y lo traduce a un texto equivalente en lenguaje máquina.

2 tipos de traductores:

1. Compilador

Traduce todo el código de alto nivel a bajo nivel generando una versión traducida del programa. Las entradas del usuario se reciben y traducen posteriormente, dando lugar a traducciones por separado.

▲ Enlazador => Liga todas las traducciones realizadas por el compilador y se genera el archivo ejecutable.

2. Intérprete

Lee un programa fuente que traduce línea a línea en tiempo real, de forma que si hay sentencias en las que el usuario debe añadir algo, lo espera. No se genera ningún programa en lenguaje máquina.

Gramáticas

Son estructuras y leyes que proporcionan una especificación sintáctica específica de un lenguaje de programación. Forma genérica de expresar gramáticas:

$G = \{V_n, V_t, P, S\}$

$V_n \equiv$ símbolos no terminales
$V_t \equiv$ símbolos terminales
$P \equiv$ reglas de producción
$S \equiv$ axioma de la gramática (símbolo por el que siempre se empieza)

Ejemplo

$G = \{ \{S, A\}, \{0, 1\}, \{S ::= AS | A,\ A ::= 0 | 1\}, S \}$

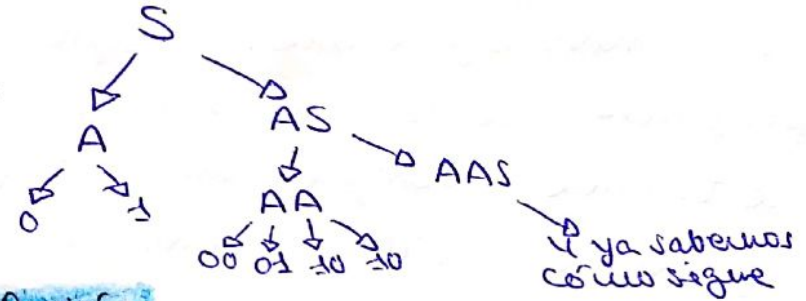
1º) $\Rightarrow_{\text{Axioma}} S \Rightarrow AS \Rightarrow AAS \Rightarrow AAA \Rightarrow 1AA \Rightarrow 10A \Rightarrow 101$

2º) $\Rightarrow S \Rightarrow AS \Rightarrow AA \Rightarrow 1A \Rightarrow 11$

Esta técnica se llama derivación

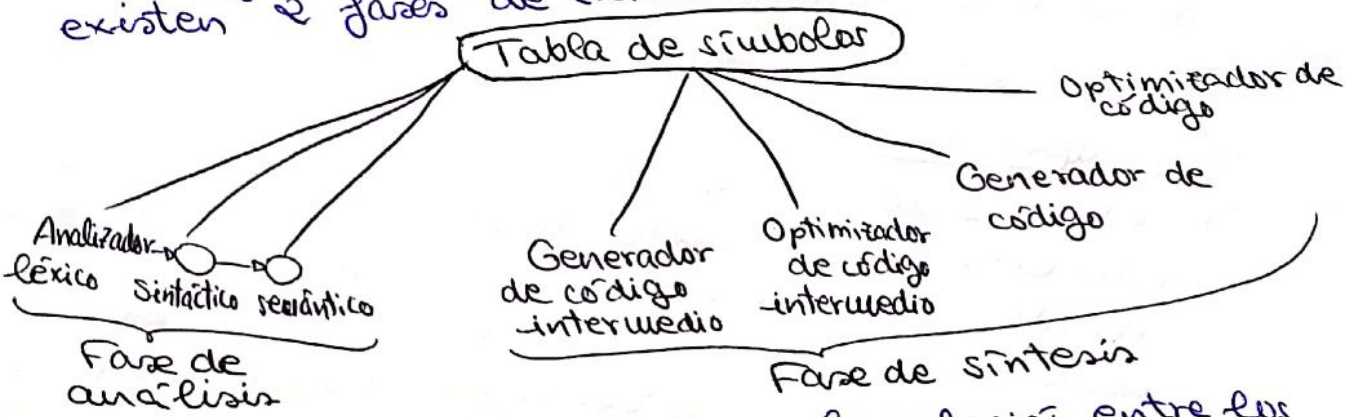
Árbol sintáctico

De aquí se deduce que esta gramática genera cualquier número en binario



Proceso de compilación

El traductor nunca traduce directamente lo que le llega, primero se verifican los errores. Por ello existen 2 fases de traducción:



La tabla de símbolos mantiene la relación entre los analizadores y generadores almacenando todas las conclusiones que se sacan en cada parte. De esta forma, los analizadores, generadores, etc. no mantienen relación y son funciones independientes.

1. Análisis léxico

Coge línea a línea el código, tomando palabra a palabra. Las palabras las clasifica en tokens. Un token es un concepto asociado a un conjunto de lexemas que tienen la misma misión sintáctica. Un lexema es una secuencia de caracteres que identifica unívocamente una palabra. Al clasificar las palabras por tokens, las introduce en la tabla de símbolos.

Ejemplo $\Rightarrow$ Token: OPERADOR — Lexemas: <=, ==, !=, ...

Si encuentra un elemento que no cuadra con ninguno, toque, avisa del error.

2. Análisis Sintáctico

Establece si la estructura en la que se organizan los tokens es correcta (lo determina ayudándose de las reglas de producción de la gramática). En resumen, analiza la validez de las secuencias de tokens. Si no hay errores, da a conocer la secuencia de derivaciones para obtener las estructuras de tokens.

3. Análisis Semántico

Detecta construcciones sin un significado correcto. Las más comunes son:

1º) Variable no declarada
2º) Asignación entre tipos incompatibles
3º) Llamada a un procedimiento incorrecto o con nº de argumentos incorrecto.

También produce warnings para que el programador sepa que podría estar ocurriendo algo no deseado.

Generación y optimización de código

Tras la fase de análisis, se crea un archivo con código en lenguaje máquina (generación) y se produce la optimización, que consiste en mejorar el código mediante comprobaciones locales a un grupo de instrucciones o a nivel global (de esto se encarga el optimizador). Así, se reducen los elementos que causan un mayor tiempo de ejecución.

Características intérpretes

- No generan archivos objeto.
- La ejecución está siempre supervisada por el intérprete.
- Las instrucciones de los bucles se analizan tantas veces como iteraciones haya (da lugar a gran tiempo de procesado).
- Solo es posible optimizar a nivel de instrucción.

Intérpretes VS Compiladores

¿Cuándo usarlos?

Intérprete

1º) Se desean obtener los resultados de la ejecución instrucción a instrucción.
2º) La velocidad no importa y el programa lo ejecuta pocas veces.
3º) Instrucciones de estructura simple.
4º) Cada instrucción se ejecuta una sola vez.

Compilador

1º) Cuando las instrucciones son más complejas.
2º) Cuando la velocidad es importante.
3º) Las instrucciones se ejecutan con frecuencia (bucles).

Modelos de memoria de un proceso

El manejo de memoria se ha hecho más sencillo al establecer 3 niveles destinados a su gestión:

Nivel de procesos → El SO se encarga del reparto de memoria entre los procesos

Nivel de regiones → La división en regiones la hace el compilado. El SO distribuye entre ellas el espacio.

Nivel de zonas → Reparto de una región entre las zonas "nivel estático", "dinámico en pila" y "heap".

Tipos de datos

Datos estáticos: Globales a todo el programa o locales a una función. Pueden ser constantes o variables, con o sin valor inicial.

Datos dinámicos: Se almacenan en pila en un registro de activación (frame). Se crean al activar una función y se destruyen al acabar esta.

Heap: Datos dinámicos controlados por el programa (como en MP).

Ejemplo ⟹

```
int a;                          } Datos
const int b=8;                  } estáticos
void funcion (int h)
  { int j; => Dato dinámico
  }
```

Ciclo de vida de un programa

Son todas las fases por las que pasa un programa hasta que se ejecuta. A partir de un código fuente, un programa pasa por una serie de fases:

1. Preprocesado. Transforma archivos .c a .i.
2. Compilación. Traduce el código preprocesado en diferentes archivos que necesitan ser unidos, archivos .i a .s.
3. Ensamblado. Unifica los archivos .s en un único .o.
4. Enlazado. Toma el archivo objeto .o, bibliotecas .h, añade código relocalizable (si es necesario) para generar un archivo ejecutable en binario interpretable por el SO.
5. Carga y ejecución. El SO ejecuta el programa cargándolo en memoria.

Compilación

Las principales acciones del compilador son:

- Generar código objeto y calcular el espacio que ocupan los diferentes tipos de datos.
- Asigna direcciones a los símbolos estáticos y resuelve las referencias de forma absoluta o relativa.
- Las referencias a símbolos dinámicos se resuelven usando direccionamiento relativo a pila o con direccionamiento indirecto para el heap.
- Genera la Tabla de símbolos e información de depuración.

Enlazado

Agrupamiento de los archivos objeto de la aplicación y las bibliotecas, resolviendo las referencias entre ellos. Una de las funciones más importantes realizada en esta etapa es el agrupamiento de módulos en regiones (es decir, se separan las variables inicializadas y no inicializadas, constantes, etc).

Tipos de enlazado:
- Externo: Visibilidad global
- Interno: Visibilidad de fichero
- Sin enlazado: Visibilidad de bloque

→ Indican si el nombre de una variable o función en otro ámbito se refiere al mismo objeto o a otro distinto.

Finalmente se produce la carga en memoria principal (con reubicación estática o dinámica) y la ejecución

Diferencias archivos objeto y ejecutable

Los archivos objeto son resultado de la compilación y los ejecutables del enlazado. Los ejecutables cuentan con la primera instrucción que se cargará en el PC en su cabecera, cosa que los archivos objeto no.

Los ejecutables se dividen en:

Cabecera → Cuenta con un número mágico que indica qué tipo de ejecutable es. Posee también el PC, tamaño del código, etc.

Secciones → Contiene el código y las variables.

Bibliotecas

Son una colección de objetos relacionados entre sí que favorecen la modularidad y reusabilidad del código.
Hay 2 tipos:

Estáticas

1. Construimos el código fuente
2. Generamos el objeto
3. Archivamos el objeto (creamos la biblioteca)
4. Utilizamos la biblioteca

Su principal inconveniente es: el desperdicio de disco y memoria principal ya que el código de la biblioteca está en todos los ejecutables que la usan, adquiriendo gran tamaño. Además, si esta sufre alguna modificación, es necesario recompilar los programas para evitar errores.

Dinámicas

Se integran con los procesos que las usan en tiempo de ejecución, por ello se realiza antes la reubicación de módulos. Así se evitan los problemas de memoria y recompilación de código que había con las estáticas.
El archivo correspondiente a una biblioteca dinámica se caracteriza por:
Contiene información de reubicación.
Contiene una tabla de símbolos.
En la cabecera no se almacena información de punto de entrada.

* El uso de bibliotecas dinámicas requiere un enlazador dinámico, que las carga y monta durante la ejecución del programa.