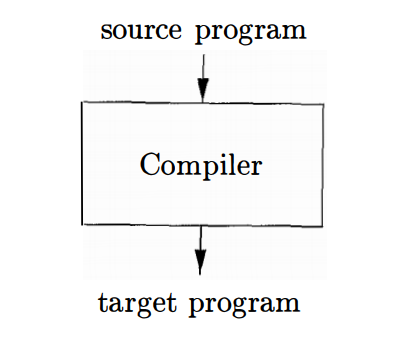
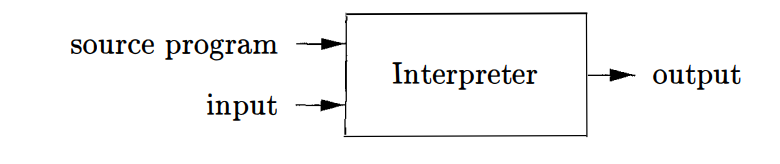
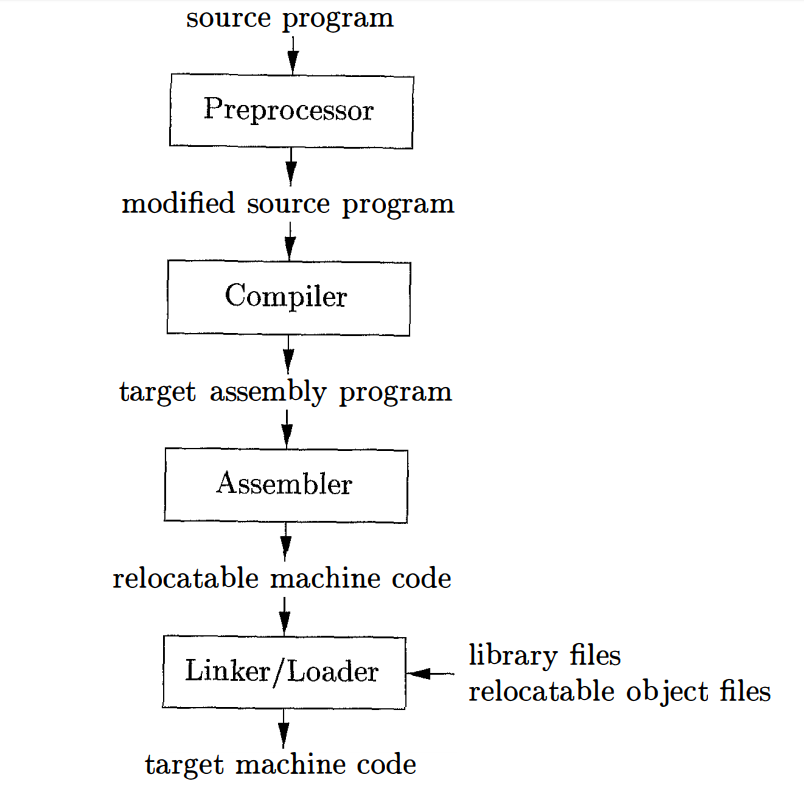
Compiladores Capitulo 1 Resumen

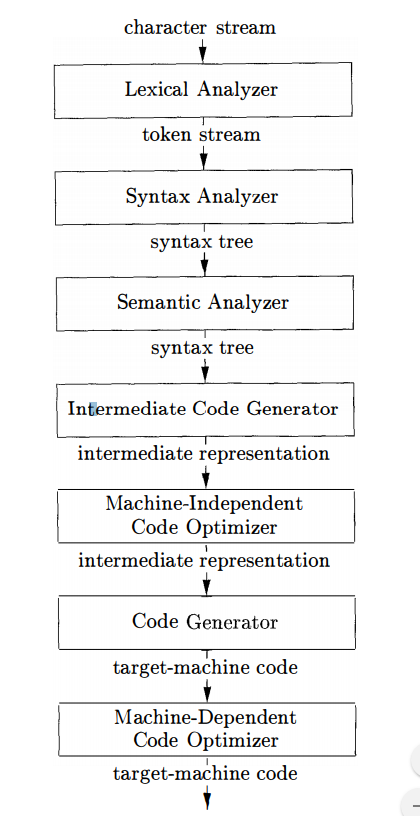
Para que un programa pueda correrse por una computadora, necesita pasar por un proceso de tratamiento que convierte de un lenguaje (generalmente lenguaje de programación) a otro lenguaje (lenguaje de maquina). Una de las principales tareas del compilador es la de reportar cualquier error que se encuentre durante el proceso de traducción.

Un interpretador es un tipo de procesador de lenguaje. Un programa en lenguaje maquina previamente compilado es mucho más rápido que un interpretador a la hora de mapear resultados. Pero el interpretador es mejor a la hora de dar diagnósticos de error que un compilador.



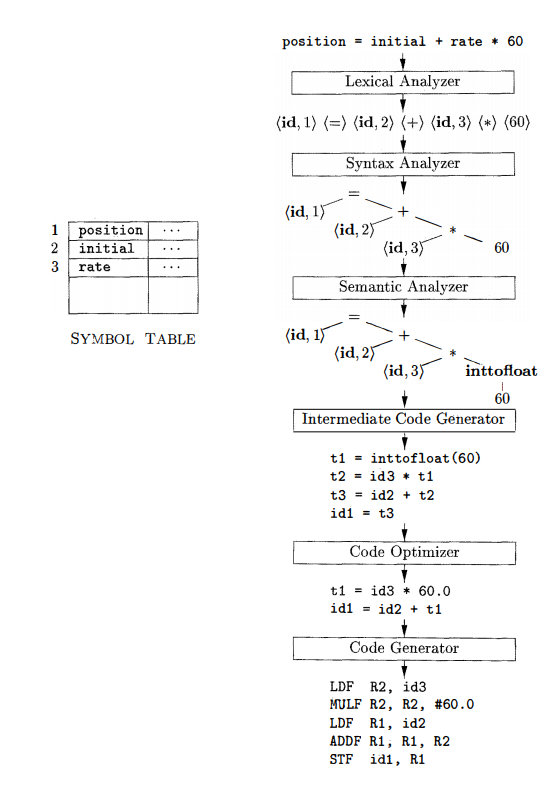
En algunos casos se requieren de programas adicionales para que se cree un programa ejecutable. Un **preprocesador** es un programa que divide el **Source Program** en archivos separados. De ahí pasa al **compilador** que produce un programa en **lenguaje de ensamblado** (por su facilidad al producirse y debuggear). Luego este programa se procesa por un **ensamblador** que como output genera el lenguaje máquina.

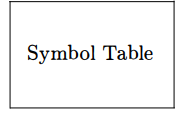
En algunos programas grandes, se suelen generar varios archivos. Aquí es donde entra el **linker** que soluciona las direcciones de memoria que hagan referencia a otros archivos y librerías necesarias para que al final, el **loader** cargue todos estos archivos en memoria para su ejecución.

**Estructura de un compilador**

* **Análisis:** También llamado front end, rompe el programa origen en partes que se les asigna una estructura gramatical en ellos para crear una representación intermedia del programa de origen. Si se detecta que hay alguna sección tanto sintáctica como semánticamente errónea, arroja mensajes de error referentes a estos para su corrección. Ademas de guardar información importante del programa de origen en una estructura llamada **tabla de símbolos** que se pasa a la etapa de síntesis.
* **Síntesis:** Se construye el **target Program** a partir de la representación intermedia y de la tabla de símbolos.

**Análisis Léxico:** Lee la serie de caracteres que componen el programa origen y los agrupa en secuencias llamadas lexemas. Para cada lexema producen un token de la forma **<token-name, attribute-value>** y los pasa a la siguiente fase.



**Análisis sintáctico:** También llamado **parser**, utiliza los tokens producidos por el análisis léxico para crear una representación en forma de árbol en orden que se deben de ejecutar las operaciones.

**Análisis semántico**: Revisa consistencia semántica con respecto a la definición del lenguaje. Detecta información de tipos de datos **(type checking)** y que no haya errores de este tipo. Dependiendo de la definición del lenguaje se pueden por ejemplo permitir casos como este: un operador aritmético puede ser aplicado tanto a un par de flotantes como de enteros, pero en el caso de tener un flotante y un entero, el compilador tiene que convertir el entero a punto flotante.

**Generación de Código intermedio:** En esta etapa se produce un código intermedio fácil de generar (a partir del árbol) y fácil de traducir a nuestro target machine.

**Optimización de código:** es la etapa donde se intenta mejorar el código intermedio generado anteriormente para que el código target sea mejor.

**Generación de Código:** En esta etapa se toma el código intermedio y se traduce a lenguaje de maquina (lenguaje de ensamblado) además de asignar los registros correspondientes de nuestra tabla de datos que se utilizó en el código.

**Evolución de los Lenguajes de Computadora:**

* 1940 se programaban las primeras computadoras con secuencias de 0s y 1s. Solo podían mover datos de un lado a otro, agregar los contenidos de dos registros, comparar valores, etc. Era una programación lenta, tediosa y tendiente a errores.
* 1950s se desarrollan los primeros lenguajes de ensamblador. Las instrucciones que se agregaban eran atajos que utilizaban para secuencias instrucciones que solían usar con frecuencia. Se desarrolló **Fortran** para computación científica, **Cobol** para procesamientos de datos de negocios, **Lisp** para computación simbólica.

Los lenguajes de **primera** generación son los **lenguajes de máquina**, los de **segunda** generación son los lenguajes de **ensamblado**, los de **tercera** generación (de alto nivel) son lenguajes como **Fortran, Cobol, Lisp, C, C++. C# y Java**. Los de **cuarta** generación son lenguajes para aplicaciones especificas como **SQL** para bases de datos, los de **quinta** generación son lenguajes aplicados a lógica.

Los lenguajes **imperativos** como C, Java, etc. Especifican como la computación se tiene que realizar.

Los lenguajes **declarativos** son los que se especifica que operación se va a realizar.