Intérpretes

Tipos de Intérpretes

Intérpretes puros:

Son los que analizan y ejecutan sentencia a sentencia todo el programa fuente y siguen el modelo de interpretación iterativa. El principal problema de este tipo de intérpretes es que, si a mitad del programa fuente se producen errores, se debe de volver a comenzar el proceso.

Intérpretes avanzados:

También llamado intérpretes normales, estos incorporan un paso previo de análisis de todo el programa fuente; generando posteriormente un lenguaje intermedio que es ejecutado por ellos mismos. Cuando hay errores sintácticos no pasan de la fase de análisis. Permiten realizar un análisis detallado del programa fuente (comprueban tipos, optimizan instrucciones, etc.).

Intérpretes incrementales:

Estos compilan aquellas partes estáticas del programa en leguaje fuente, marcando como dinámicas las que no puedan compilarse. En el tiempo de ejecución, el sistema compila algunas partes dinámicas o recompilarlas si han sido modificadas. No producen un código objeto independiente, sino que acompañan el sistema que permite compilar módulos en tiempo de ejecución al código objeto generado.

Evaluadores parciales:

Se utilizan cuando muchos programas contienen dos tipos de datos de entrada. Existen una serie de datos de entrada que son diferentes en cada ejecución mientras que otros datos no varían de una ejecución a otra. El primer conjunto se conoce como datos de entrada dinámicos y el segundo datos de entrada estáticos. Su principal ventaja es la eficiencia; si se conoce que un programa va a ejecutarse muchas veces con un conjunto de datos estáticos pero diferentes datos dinámicos, será más eficiente evaluar parcialmente el programa para obtener el programa estático y ejecutar luego el programa dinámico. Los evaluadores parciales realizan un complejo análisis del programa fuente para detectar que el proceso no genere un bucle infinito. También tienen la posibilidad de generar compiladores a partir de intérpretes.

Compiladores “Just in Time (JIT)”

Estos compilan el código a código nativo justo en el momento en que lo necesita el programa que se esté ejecutando. Sus principales ventajas son: los programas grandes contienen porciones de que no son ejecutados en una ejecución típica del programa; debido a que la compilación JIT sólo traduce aquellas porciones de código que se necesitan y se evita compilar código innecesario. Y los sistemas tradicionales realizan la compilación de todo el código antes de la ejecución.

Compilación Continua:

El sistema mezcla el proceso de compilación a código nativo con el proceso de interpretación. Para ello, el sistema tiene dos módulos: uno de interpretación de los códigos de bytes y otro de compilación de códigos bytes a código nativo. El código contiene una mezcla de código fuente y código nativo del programa; el módulo compilador traduce las unidades de compilación a código nativo y este código nativo se deja disponible al intérprete; el módulo interprete se responsabiliza de la ejecución del programa, interpreta el código fuente; y, por último, el monitor se encarga de coordinar la comunicación entre los dos módulos anteriores.

INTÉRPRETE	VENTAJAS	DEVENTAJAS
PURO	Permiten la ejecución de largos programas en ordenadores de memoria reducida.	Si se producen errores, se deberá empezar de nuevo el análisis.
AVANZADO	Si hay errores sintácticos se detiene el análisis.	--------
INCREMENTALES	Mayor eficiencia de ejecución.	--------
EVALUADORES PARCIALES	Evalúa dos tipos de datos, estáticos y dinámicos. Se pueden generar compiladores a partir de la misma.	--------
COMPILADORES JIT	Compila todo el código antes de ser ejecutado y solo compila el código que es útil.	--------
COMPILACIÓN CONTINUA	Traduce a código nativo y luego lo interpreta.	--------

Herramientas para generar analizadores

Lex:

Es un generador de analizador léxico, que sirve para generar los token para la siguiente fase del analizador. Permite asociar acciones descritas en C, a la localización de las Expresiones Regulares que se hayan definido. Se apoya en una plantilla que recibe como parámetro, y se debe diseñar con cuidado. Actúa como un autómata que localiza las ER que se le describan y una vez reconoce la cadena representada, ejecuta el código asociado a esa regla. Se utiliza en lenguaje Pascal y C.

Flex:

Es una herramienta para generar escáneres: programas que reconocen patrones léxicos en un texto. Flex lee los ficheros de entrada dados, o la entrada estándar si no se le ha indicado ningún nombre de fichero, con la descripción de un escáner a generar. Se utiliza en lenguaje C, C++ y Java (JFlex).

ParseGenerator:

Es una IDE para generadores AYACC y ALEX los cuales son clones de Yacc y Lex.

Yacc:

Es un programa para generar analizadores sintácticos. Sus siglas significan Yet Another Compiler Compiler. Genera el analizador basado en una gramática analítica. Se utiliza en lenguaje Pascal y C

Bison:

Es un generador de analizadores sintácticos de propósito generar que convierte una descripción gramatical para una gramática independiente del contexto en un programa en C que analice esa gramática. Es utilizado en un amplio rango de analizadores de lenguajes. Es la versión mejorada de Yacc y se utiliza en lenguaje C, C++ y Java.

YAY:

Es un generador de analizadores sintácticos ascendentes similar a Yacc. Este soporta gramáticas LALR(2). Es utilizado en lenguaje C.