Trabajo Práctico Número 2
Teoría de la Computación

Ulises C. Ramirez

28 Abril 2018

1 ¿Qué es un compilador?

A grandes rasgos, un compilador es un programa que lee un programa escrito en un lenguaje (lo podemos denominar como el lenguaje *fuente*), y lo traduce a un programa equivalente en otro lenguaje (al cual podemos denominar lenguaje *objeto*). Como parte importante de este proceso de traducción, el compilador informa a su usario de la presencia de errores en el programa fuente.

Estos compiladores pueden clasificarse como: de una pasada, de múltiples pasadas, de carga y ejecución, de depuración o de optimización, dependiendo de como hayan sido construidos o de qué función se supone que realizan.

2 ¿ Qué es un intérprete de lenguaje de compilación?

Como se especifica en [1], un *intérprete* es otro tipo común de procesador de lenguaje. En vez de producir un programa destino como una traduccion, el intérprete nos da la apariencia de ejecutar directamente las operaciones especificadas en el programa de origen (fuente) con las entrdas proporcionadas por el usuario.

3 Enumere las diferencias entre un compilador y un intérprete

A continuación se enumeran las diferencias solicitadas a modo de un cuadro comparativo sencillo, la informacion ahí detallada se extrajo de [1].

Compilador	**Intérprete**
Lee un programa fuente	Lee un programa fuente, y entradas
Realiza una serie de procesos estudiados en el TP1 y genera un programa destino	Interpreta las entradas que recibio y genera una salida
Acepta entradas del usuario unicamente una vez pase por el proceso de compilacion	las entradas se interpretan junto con el codigo del programa

4 Enumere las ventajas de Compiladores sobre Intérpretes y las de Intérpretes sobre Compiladores

4.1 Ventajas de Compiladores sobre Intérpretes

[1] [2]

- El programa destino producido por un compilador es, por lo general, más rapido que un intérprete al momento de asignar las entradas a las dsalidas
- Una vez compilado, la computadora es capaz de leer el lenguaje maquina y no tener que interpretar linea a linea las sentencias del programa.
- Al tener que ser inspeccionados en la etapa de compilacion por el compilador, a estos se les puede descubrir errores relacionados con el checkeo de tipeo estatico [3]. El checkeo de tipeo estatico puede ser considerado una optimizacion, ya que un compilador puede proar que si un programa está bien tipeado, entonces no necesita emitir *chequeos de seguridad dinámicos*

4.2 Ventajas de Intérpretes sobre Compiladores

[1] [2]

- Este por lo regular puee ofrecer mejores diagnosticos de error de un compilador, dado a que ejecuta un programa funte, instrucción por instrucción.
- Un intérprete no posee proceso intermedio para la obtencion de un programa fuente, por lo que si se tiene un codigo muy grande, este lo ejecutará linea a linea.
- Facilita la depuración y prueba de código dado a que no debe pasar por un proceso de compilacion cada vez que se ejecute.
- Se tiene la posibilidad de leer el código fuente sin tener que decompilar el paquete.

5 ¿En qué situación eligiría un lenguaje compilado y en qué situación un lenguaje interpretado?

En los tiempos actuales y con la tecnología disponible, esta eleccion se basa en un aspecto fundamental desde mi pundo de vista. En primera instancia podemos decir que diferencia en rendimiento entre ambos programas únicamente se verá cuando el ecosistema en el que se encuentre el programa sea muy demandante, ejemplo: Twitter, Facebook, Google. En estos entornos la ejecución de los programas afecta en gran medida al usuario final y podría ser la diferencia en que uno elija un sistema u otro. En casos muy extremos como estos, seria conveniente elejir algun lenguaje que funcione a bajo nivel y que deba pasar por una etapa de compilacion como lo puede ser C, en cambio, valiéndome del argumento mencionado al principio, acerca de las tecnologías disponibles actualmente, para lugares menos demandantes (No por eso significa que sean entornos de una escala excesivamente pequeña) seria conveniente elegir un lenguaje interpretado dado a que los tiempos de compilacion serian inexistentes y las pruebas podrán hacerse sin tener que pasar por la etapa mencionada.

Sin embargo, no todo tiene por que ser blanco o negro, a lo que me refiero con eso es que, es posible que se decida utilizar en las secciones de codigo que necesitan de un nivel de respuesta excesivamente alto, algun lenguaje que se ajuste a esas necesidades como ser bajo nivel, y en las partes menos demandantes utilizar lenguajes de mas alto nivel y con capas de abstracción que permitan que la productividad sea mayor en el equipo de desarrollo. Como ejemplo, podemos hablar de un backend en alguna aplicación cliente-servidor, en el cual tendremos que la parte "pesada y demandante" la va a ejecutar el servidor, por tanto esta será la parte que estará desarrollada en C o Assembler, por ejemplo, y luego el cliente, que basicamente estará funcionando como una interfaz para la carga, puede utilizar lenguajes de mucho mas alto nivel como pueden ser C++, Java, Python, Ruby etc, que contienen capas de abstraccion que facilitan su comprensión y manejo.

6 Ejemplos de lenguajes compilados e interpretados

Se eligieron los ejemplos nombrados a continuacion debido a la afinidad de algunos de ellos con la bibliografia consultada. Dado que la misma nombra constantemente a lenguajes como C, C++, COBOL, etc. se decidió incluirlos en los ejemplos y clasificarlos.

6.1 Ejemplos de lenguajes Compilados

Como ejemplo de lenguajes compilados podemos mencionar los siguientes:

- C++
- C
- COBOL
- Fortran
- Java
- Pascal

6.2 Ejemplos de lenguajes Interpretados

Dada la amplia demanda en los tiempos actuales de personas que buscan aprender el diseño de aplicaciones con arquitectura web, podemos hacer mención específicamente de lenguajes que son, justamente, interpretados por el motor del navegador y descifrado para obtener elementos del **DOM** y asi visualizar lo que se programa con lenguaje de tipo etiquetado y de estilos (HTML, CSS), a continuación se nombran como ejemplos de lenguajes interpretados algunos de los mas famosos:

- PHP
- Javascript
- Ruby (Menciono a esto por su popular framework para desarrollo web *Ruby on Rails*)
- Python (Popular dentro de lo que es scripting, aunque tambien popular en desarrollo web con su framework *Django*)

Referencias

[1] Alfred V. Aho; Monica S. Lam; Ravi Sethi; Jeffrey D. Ullman *Compiladores: principios, técnicas y herramientas*, Segunda Edición. Pearson Educación, México, 2008. isbn: 978-970-26-1133-2.

[2] Alfonseca Moreno, M.; De la Cruz Echeandía, M.; Ortega de la Puente, A.; Pulido Cañabate, E *Compiladores e Intérpretes: teoría y práctica*. Pearson Educación, S.A, Madrid, 2006. isbn 10: 84-205-5031-0; isbn 13: 978-84-205-5031-2

[3] Wikipedia, Type system *Static type-checking*.