**Universidad Tecnológica Centroamericana**

**UNITEC**

**Tarea 1**

**Estudiante:**

|  |  |
| --- | --- |
| Laurenn Alecxandra Cruz | 11211190 |

**Compiladores I, Sección: 435**

**Catedrático: Carlos Vallejo**

**Tegucigalpa, M.D.C., 24 de enero de 2016**

**Historia de los compiladores y su impacto en la actualidad**

**Introducción**

Un compilador según Singh (2009) es un programa complejo de computadoras escrito en algún lenguaje. Inicialmente, los compiladores eran escritos en assembler. El primer compilador “self-hosting” (se le denomina a un compilador “self-hosting” cuando es contruido con el mismo lenguaje que compila, un lenguaje de alto nivel contrario a los compiladores construidos con assembler) fue construido por Hart and Levin en el MIT en el año de 1962 para el lenguaje Lisp.

Otras historias alternas situan la creación del primer compilador en el año 1952, de la mano de Grace Hooper. Así como sustenta en su artículo Engel (2014), Hooper desarrollo un lenguaje llamado A-0 (Arithmetic Language version 0) para la UNIVAC I. Algunos teóricos afirman que A-0 es considerado más un linker que un compilador propiamente dicho.

Si somos más pragmaticos, el primer compilador completo data del año 1957 en la cuna de IBM, como se menciona en la página oficial de la compañía (2011):

“Desde su creación en 1954, y su salida al mercado en 1957 como el progenitor del software, Fortran (FORMula TRANslator) se convirtió en el primer lenguaje de computadoras, ‘ayudó a abrir las puertas a la computación moderna’ y bien podría considerarse el software que más influenció la historia […]”.

En 1958 fue creado el primer compilador de ALGOL 58 de la mano de Friedrich L. Bauer, Hermann Bottenbruch, Heinz Rutishauser y Klaus Samelson. Con el pasar de lo años, existieron problemas en cuanto a la arquitectura donde era ejecutado un compilador, así nacen los compiladores “cross-platform” liderados por COBOL en 1960. Como una pieza histórica en The National Museum of American History (2016) se encuentra una copia del primer programa desarrollado en COBOL que se ejecutó en dos computadoras de diferentes fabricantes en su presentación en 1960.

Para diferenciar las distintas épocas y avances que tuvieron los compiladores a lo largos de los años, se categorizan en los subtemas de “self-hosting”, generadores de gramáticas libres de contexto, JIT Compilers: Representación de código intermedio.

**Self-hosting**

Existen muchos lenguajes que cumplen esta característica actualmente: Ada, BASIC, C, C++, C#, CoffeeScript, Go, Haskell, Java, Lisp, Python, Scala, Visual Basic, por mencionar unos pocos. La idea de estos compiladores, inicialmente propuesta en 1962 por Hart and Levin con la creación de Lisp, es la de ser capaz de compilar su propio código fuente. Incluido en kernels, assemblers, intérpretes de línea de comandos y controladores de versiones de software.

**Generadores de gramáticas libres de contexto**

Existen diversos componentes comunes para modularizar un compilador, uno de ellos es el parser. Como menciona O’Donnell (2008) la función principal de un parser es la de crear una representación interna para el compilador, comunmente denominada árbol de parseo, la cual es usada por el generador de código para producir código objeto. Otra de las funciones del parser es la detección de errores sintácticos en el código fuente.

Los parsers pueden ser escritos a mano, o generados por un generador de parsers. La unidad de éstos son las gramáticas libres de contexto, que ayudan a fabricar un parser en pequeños bloques que representan sencillas leyes gramaticales que el parser debe diferenciar. Esta arquitectura fue introducida por ALGOL en sus inicios.

Las gramáticas libres de contexto son suficientemente simples como para crear algoritmos de parseo muy eficientes, inclusive, si un lenguaje de programación reconocido por un compilador es representado por un subconjunto de los lenguajes de contexto libre, se pueden encontrar algoritmos considerablemente más eficientes en comparación con su conjunto universo.

**JIT Compilers: Representación de código intermedio**

Existen otro tipo de compiladores que para poder adaptarse a múltiples plataformas, fabricantes y arquitecturas tuvieron que relegar su compilación hasta el tiempo en que el programa es ejecutado (Visión general del compilador JIT, 2015). Actualmente lenguajes de programación con compiladores muy famosos como Java utilizan esta técnica para poder agilizar la lectura de código intermedio (bytecode en este caso). La técnica fue introducida en 1996 de la mano de Oracle.

**Influencia en la actualidad**

Uno de los temas importantes de los creadores de compiladores actualmente en el uso de multinucleo en el momento de la compilación. Como menciona Langkamp y Böhner (2011) los procesos de compilación son complejos, de intesivo uso de recursos y comunmente caros en tiempo de compilación. Si se utilizaran varios núcleos puede aprovecharse todo el potencial e inclusive, optimizar el código objeto de maneras que un ordenador convencional o un compilador convencional no es capaz de aprovechar.

Compiladores de la era móvil como el ART integrado en el sistema operativo móvil Android realizan lo que ellos denominan Ahead-of-Time (AOT) compilation. Google (2015) introduce este tipo de compilación para mejorar la ejecución de una app en un dispositivo con Android. En el momento de la instalación una herramienta denominada *dex2oat* recibe de input la aplicación en código intermedio y la compila para la arquitactura del dispositivo donde se ejecuta el ART.

Los editores de texto actuales, inclusive los procesadores de texto y asistentes por voz como Siri y Google Now, están construidos a partir de la teoría de compiladores y utilizan diversos pasos de ésta para interpretar lo que el usuario escribe o habla.

**Conclusiones**

La historia de cómo los compiladores fueron elaborados y sus precursores está poco documentada y se requiere indagar mucho en los libros antiguos para presentar evidencia sustancias y noticias que puedan indicarnos el origen de las teorías e ideas detrás de un compilador.

Así mismo las etapas en que un compilador se fue desarrollando, haciendo más complejo y optimizando van desde los “self-hosting” que no requerían un software preexistenten para construirlo, pasando por los generadores que simplificaron las tareas tediosas en los módulos de un compilador como el parser, hasta la optimización en corrida de la mano de los JIT compilers que ayudan hoy a los móviles, por ejemplo, a ejecutar aplicaciones rápidamente.

Hoy en día nos vemos rodeados por muchas apps y software en general que utiliza aunque sea parte del proceso de compilación para llevar sus tareas acabo.

# **Bibliografía**

Cook, A. (4 de marzo de 2014). *Cunningham & Cunningham, Inc.* Obtenido de Dynamic Typing: http://c2.com/cgi/wiki?DynamicTyping

Engel, K. (2014). *Admiral “Amazing Grace” Hopper, pioneering computer programmer* . Recuperado el 23 de Enero de 2016, de Amazing Women in History: http://www.amazingwomeninhistory.com/amazing-grace-hopper-computer-programmer/

Google. (2015). *ART and Dalvik*. Recuperado el 24 de Enero de 2016, de Android: https://source.android.com/devices/tech/dalvik/

IBM. (2011). *FORTRAN The Pioneering Programming Language*. Recuperado el 23 de Enero de 2016, de IBM at 100: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/fortran/

Langkamp, T., & Böhner, J. (27 de Junio de 2011). *Influence of the compiler on multi-CPU performance of WRFv3*. Recuperado el 24 de Enero de 2016, de Geoscientific Model Develpment: http://www.geosci-model-dev.net/4/611/2011/gmd-4-611-2011.html

O'Donnell, M. (19 de Febrero de 2008). *Topic 3: Syntactic analysis (LR).* Recuperado el 23 de Enero de 2016, de Compilers 2008: http://arantxa.ii.uam.es/~modonnel/Compilers/03\_1\_Parsing\_Intro.pdf

Singh, R. (2009). Introduction to Compiler. En R. Singh, *Design and Implementation of Compiler* (pág. 42). New Age International.

The National Museum of American History. (2016). *COBOL*. Recuperado el 23 de Enero de 2016, de Smithsonian: http://americanhistory.si.edu/cobol/introduction

*Visión general del compilador JIT*. (2015). Recuperado el 24 de Enero de 2016, de IBM Knowledge Center: https://www-01.ibm.com/support/knowledgecenter/SSYKE2\_7.0.0/com.ibm.java.aix.71.doc/diag/understanding/jit\_overview.html