|  |  |
| --- | --- |
| Evolución y Uso de los Compiladores  Compiladores I | UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA  UNITEC  Horacio Galdamez  No. Cuenta 11241022  Ing. Carlos Enrique Vallejo Mejía |

Contents

[Introducción 2](#_Toc441522190)

[¿Qué es un compilador? 3](#_Toc441522191)

[Tipos de Compiladores 3](#_Toc441522192)

[Historia y Evolución de los Compiladores 4](#_Toc441522193)

[Código Intermedio 7](#_Toc441522194)

[Usos actuales de los compiladores 8](#_Toc441522195)

[Conclusiones 10](#_Toc441522196)

[Bibliografía 11](#_Toc441522197)

# Introducción

Hoy en día, los desarrolladores de software contamos con muchas herramientas que nos facilitan el trabajo. Herramientas que surgieron a raíz de una necesidad y que se crearon por medio de estudio e innovación. Gracias a la demanda de procesos automatizados, y la facilidad del almacenamiento digital, la industria del software ha crecido de manera extraordinaria en cuestión de años. Cada día, una nueva necesidad para tener un sistema automatizado y digital nace, y para resolverla, contamos con una gran variedad de soluciones distintas para satisfacerla. ¿Queremos tenerlo web?¿Almacenar nuestra data en la nube?¿Automatizar el sistema de marcaje de asistencias?¿Generar reportes en cuestión de segundos?¿Proyectar nuestras ganancias basados en el lanzamiento de un nuevo producto? Todas estas interrogantes surgen, y para todas existen miles de posibilidades viables y sencillas de aplicar.

Sin embargo, ésta no siempre ha sido la situación del mundo. Antes, el humano no contaba con ningún proceso automatizado que le ayudara a ubicar una dirección a través de su teléfono celular. Un estudiante no podía jugar en línea en su computadora portátil con sus amigos en otro continente al mismo tiempo. Esto es porque antes, no era tan fácil programar, y el arte de programar era una tarea exclusiva a un número extremadamente limitado de profesionales. Antes, todo era todavía más tedioso, y se requería de mucha más atención y concentración para realizar las instrucciones más sencillas.

Gracias al conocimiento y visión de los científicos del pasado, la humanidad había empezado a descubrir el potencial de las computadoras, y es aquí cuando la sociedad busca los sistemas computacionales, luego los busca para uso personal, y mejor aún, busca una manera sencilla para permitir que cualquiera escriba sus propias líneas de código.

Existieron muchos eventos para lograr los primeros pasos para lograr que la escritura de código sea más sencilla a lo que fue inicialmente. Aquí es cuando entra el compilador. En este ensayo, aprenderemos sobre todos los intentos y aportes del ser humano que llevaron a la gran era tecnológica que conocemos hoy en día.

# ¿Qué es un compilador?

Los compiladores son programas de computadora que traducen de un lenguaje a otro. Un compilador toma un “lenguaje de alto nivel” como entrada y produce otro equivalente llamado “lenguaje de bajo nivel”. Durante el proceso de compilación, se producen errores que posteriormente son enviados al programador, quien es el encargado de solucionar los ya mencionados, y repetir el proceso de compilación, hasta que no se produzca ningún error. (Serna-Pérez, 2016)

Un compilador es un traductor, y éste es un programa que toma como entrada un lenguaje fuente, y produce un lenguaje objeto o destino. (Llatas)

La utilidad indispensable de un compilador recae en el hecho de que cualquier programa escrito en un lenguaje de programación de alto nivel debe ser traducido a código objeto antes de que pueda ser ejecutado, para que todos los programadores que usen ese lenguaje usen un compilador o un intérprete. Es por esto que los compiladores son muy importantes para los programadores, y cualquier mejora que se le haga a un compilador, es una mejora que se ve reflejada en todos los programas que lo utilicen. (Historia de la Construcción de los Compiladores, 2015)

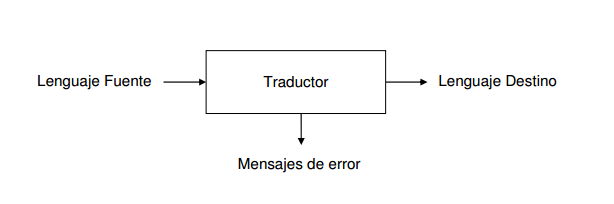


Figure 1Proceso de Traducción de Código

## Tipos de Compiladores

* Ensamblador
* Compilador Cruzado
* Compilador con Montador
* Autocompilador
* Metacompilador
* Decompilador

## Historia y Evolución de los Compiladores

A finales de los **40**s, las primeras computadoras digitales salieron a flote, y fue necesario implementar un lenguaje con el que indicábamos a la computadora a realizar cálculos. Éste lenguaje se llamaba “Lenguaje de Máquina”. Éste lenguaje representaba secuencias de códigos numéricos.

Sin embargo, éste lenguaje era muy difícil y complicado para seguir su lógica y darle mantenimiento, por lo que se estudiaron otras formas de interactuar con la computadora, y se llegó al “lenguaje ensamblador”, lenguaje cuyas instrucciones y localidades de memoria son formas simbólicas. El funcionamiento básico de este lenguaje era aquel de convertir las largas secuencias de números del lenguaje máquina y convertirlas a formas simbólica más cortas y fáciles de recordar. Ejemplo:



Figure 2 Lenguaje Máquina

Se convierte a:



Figure 3 Lenguaje Ensamblador

A pesar de ésta gran mejora al lenguaje y su legibilidad, el lenguaje ensamblador seguía siendo difícil de mantener. Es aquí cuando los desarrolladores entienden la necesidad de crear un lenguaje que permita escribir programas de una manera similar a la notación matemática que ya se conocía, como “X = 2”. A partir de esto, en 1950, se comienzan a hacer los primeros trabajos de traducción de fórmulas aritméticas a código máquina. Ya se empieza a mencionar el término “compilador”. (Serna-Pérez, 2016)

En **1945**, John Von Neumann concibe dos ideas que cambiaron el curso de la programación de computadoras:

* El hardware simple debería ser controlado por instrucciones complejas, y que el programa sea intercambiable, sin necesidad de cambiar el hardware.
* Los programas computacionales deberían ser particionados en pequeñas porciones de instrucciones que pueden saltar a cualquier otra parte del programa, lo que permitió la creación de las instrucciones IF, ELSE, y LOOP. (Modern Programming Timeline, 2016)

En **1949**, Short Code, el primer lenguaje de programación, fue desarrollado, basado en las ideas de Von Neumann. Sin embargo, el programa binario debía ser traducido de manera manual, lo que representaba un proceso demasiado tedioso y complicado. (Modern Programming Timeline, 2016)

En **1951**, Grace Hopper inicia el desarrollo de un compilador para Shortcode, cuya función era automatizar el código a binario.

IBM asigna un grupo de trabajo, liderado por John Backus, para traducir código máquina a las fórmulas matemáticas que contenían la notación que todos conocían. El proyecto resultó siendo un éxito total, pues a partir de él, en **1957**, nace FORTRAN (FORmulae TRANslation), el cuál realizaron 18 personas por más de un año. (Serna-Pérez, 2016). FORTRAN se conoció como un lenguaje de alto nivel, y su invento marca un gran avance para la computación, pues ahora se pueden escribir instrucciones con una notación más parecida a la que la gente estaba acostumbrada, lo que lleva a un mantenimiento más sencillo debido a su sencilla legibilidad.

Cabe mencionar la curiosidad del hecho de que FORTRAN fue un compilador creado por pura intuición, y no existió una teoría formal detrás de su construcción. El proceso de desarrollo era aquel en el que se iban superando retos a medida se iban presentando. (Serna-Pérez, 2016)

En **1968**, se empieza a trabajar en LISP, un lenguaje relacionado al desarrollo de la inteligencia artificial. La primera versión de ALGOL se lanza.

En **1960**, un equipo formado por fabricantes de computadoras y miembros del pentágono desarrolla COBOL (Common Business-Oriented Language). Los esfuerzos iniciales para desarrollar COBOL se enfocaron en la lectura fácil del código de programas de computadora, y también se buscaba la independencia del hardware. (Timeline of Computer History, 2016). La meta final de COBOL era que un programa pudiera correr en cualquier computadora para la cuál existiera un compilador con modificaciones mínimas.

Más adelante, se produjeron avances importantes de parte de Noam Chomsky y sus estudios acerca del lenguaje natural. Llegó a clasificar los lenguajes basados en sus gramáticas, y sus estudios derivaron en una automatización de traducción más eficiente.

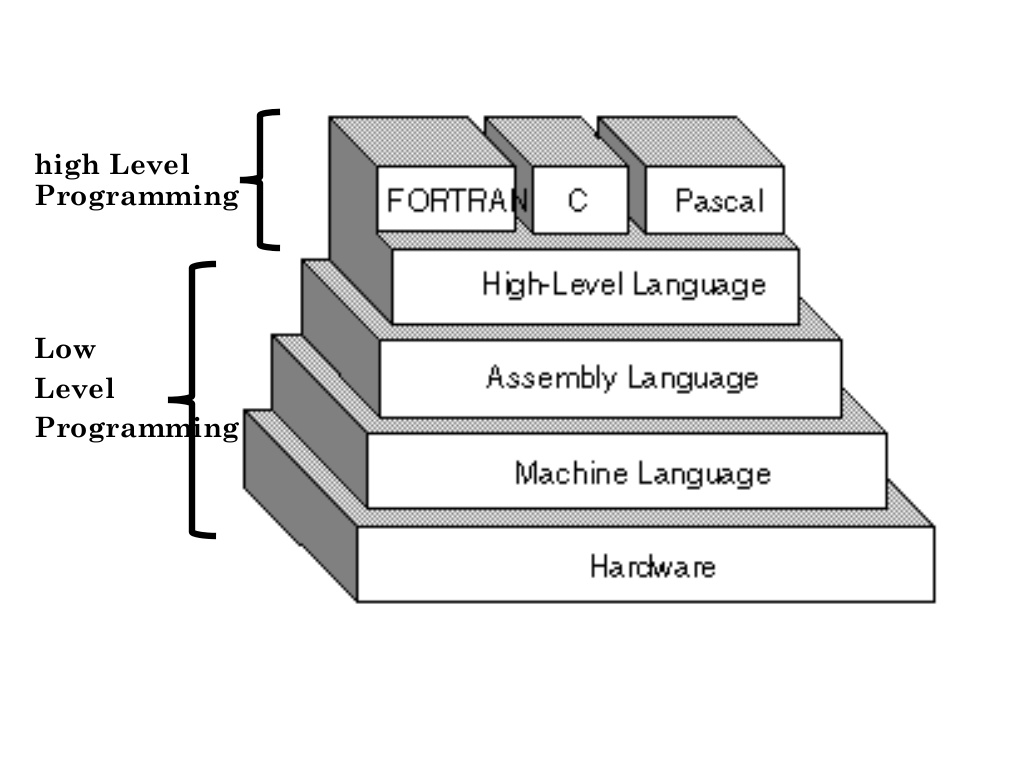


Figure 4 Lenguajes de alto y bajo nivel

Esto fue un gran paso para la computación, pues se empiezan a hacer grandes avances que permiten la traducción eficiente de código, lo que resulta en:

* Manejo más eficiente de memoria limitada, gracias a una mejor estructura en las instrucciones de los programas.
* Reducción del error humano en el proceso de traducción.
* Una nueva visión hacia mejores lenguajes de alto nivel

Después de los avances de Chomsky, surgen unos avances del proceso de compilación:

* Donald Knuth logra grandes avances en el proceso del análisis sintáctico, introduciendo e implementando el término del analizador sintáctico LR (Left to Right). Knuth describe el analizador, o “parser” como “un programa que está hecho de una familia de procedimientos que se llaman entre ellos de manera recursiva. Cada procedimiento busca una ocurrencia de algún tipo sintáctico reconocido y retorna el valor “true” o “false” dependiendo si fue exitoso o no.” (Knuth, 1971). Knuth también describe el uso de tres punteros utilizados para consumir la instrucción de un programa a la que se le hará el análisis sintáctico.
* Aparecen los primeros programas que automatizan el análisis léxico y sintáctico. (Serna-Pérez, 2016)

## Código Intermedio

Niklaus Writh, el diseñador de Pascal, propone el concepto de la representación intermedia de código, separando el proceso de traducción en dos fases:

* El front-end, la fase encargada de analizar el **programa fuente** y las operaciones dependientes sólo del lenguaje fuente.
* El back-end, encargada de generar el código para la máquina objeto. (Serna-Pérez, 2016)

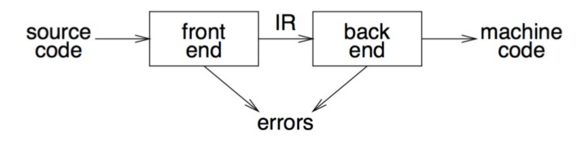


Figure 5Código Intermedio

El aporte de Writh llega a ser de suma importancia en el mundo de la programación y las computadoras, ya que este modelo de compilación nos proporciona con muchas ventajas:

1. La distinción de las partes nos ayuda a una mejor identificación de problemas, ya que trabajamos con compiladores relativamente más pequeños, donde sabremos qué errores pertencen a cuál de los dos (front-end o back-end).
2. La división de etapas permite un desarrollo más agil de los compiladores, ya que un equipo puede trabajar en el front-end y otro equipo en el back-end.
3. El front-end y back-end son independientes de su contraparte, lo que permite la fácil sustitución de uno o ambos de los compiladores para poder mejorar el proceso. (QUORA: What are advantages of frontend and backend in compilers?, 2015)

En **1980**, comienzan a proliferar las técnicas de optimización de código, y se consolida el concepto de asignación y liberación de memoria dinámica. La programación orientada a objetos es extensamente utilizada y madura. (Serna-Pérez, 2016)

Para el año **1990**, los lenguajes de programación son bastante parecidos a lo que conocemos hoy en día. Empiezan a surgir los ambientes de desarrollo, los lenguajes interpretados obtienen más fama y el código intermedio se vuelve a poner de moda. (Serna-Pérez, 2016)

## Usos actuales de los compiladores

La función principal de un compilador es aquella de traducir el programa fuente a código objeto, mientras envía errores de compilación si algo no está bien y debe ser reparado por el programador.

El compilador traduce de un lenguaje de alto nivel, como FORTRAN, C o PASCAL, a uno de bajo nivel, como el lenguaje máquina.

Para cumplir con su tarea, un compilador pasa por muchas fases de análisis, con el fin de tener un proceso modulado, y para asilar los tipos de errores posibles, para conocer más a fondo el por qué de los errores, y tener soluciones predeterminadas para cada tipo de error. Recordemos que el compilador surge a partir de una necesidad, y como tal, se busca la mejor manera de satisfacerla. Es por esto que el proceso de compilación es uno largo, pero robusto, en donde todo tipo de error aparece en cierta parte de compilación, lo que permite una depuración más sencilla de nuestro código, gracias a la manera sencilla de detectar errores. Las fases de compilación son:

**Análisis léxico:** Lectura de secuencia de caracteres de izquierda a derecha del programa fuente y agrupación de las secuencias de caracteres en unidades con significado propio, llamados tokens. (Serna-Perez, 2016)

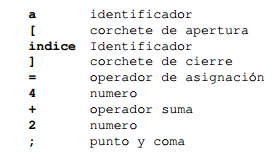


Figure 6 Identificación de componentes léxicos

**Análisis Sintáctico:** Determina si la secuencia de componentes léxicos sigue la sináxis del lenguaje y cumple con la estructura jerárquica del programa en forma de árbol. (Serna-Perez, 2016)

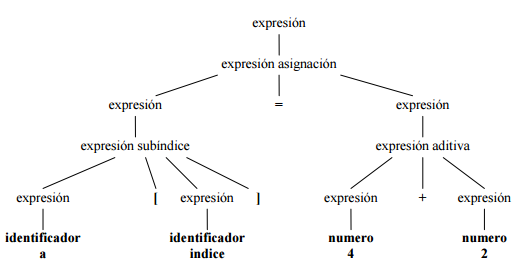


Figure 7 Árbol Semántico

**Análisis Semántico:** Realiza las comprobaciones necesarias sobre el árbol sintáctico para determinar el correcto significado del programa. Se verifican y se hace inferencia a los tipos en asignaciones y expresiones, el correcto uso de operadores, el ámbito de las variables y el llamado correcto de funciones. También verifica que los operadores utilizados concuerden con los tipos de componentes léxicos. (Serna-Perez, 2016)

**Generación y Optimización de Código Intermedio:** Consiste en la calibración del árbol sintáctico donde ya no aparecen construcciones de alto nivel. Se genera un código más fácil de traducir directamente a código ensamblador o máquina.

La etapa de optimización solo depende del lenguaje fuente y no de la máquina. Aquí se buscan eliminar sub-expresiones comunes, se identifica el código muerto, se sustiyen operaciones airtméticas, etc. Ésta suele ser una fase lenta y compleja. (Serna-Perez, 2016)

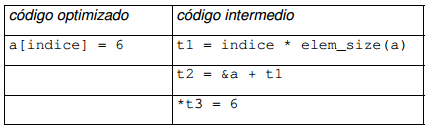


Figure 8 Código optimizado y Código Intermedio

**Generación de código objeto:** Toma como entrada la repesentación intermedia y genera el código objeto. El proceso depende de la máquina en donde se realiza, y resulta ser un proceso de rendimiento variado, ya que existen muchos casos particulares debido a la diferencia de arquitecturas existentes hoy en día. (Serna-Perez, 2016)

## Conclusiones

En resumen, un gran número de eventos tuvieron que ocurrir antes de la concepción de los compiladores. A raíz de la necesidad de contar con un lenguaje fácil de utilizar, leer y mantener, el humano ha formulado muchas soluciones para llegar al compilador de hoy en día. Desde traducir el código a ensamblador manualmente, hasta llegar a un traductor automático, y luego haciendo el tema todavía más completo, dividiéndolo en frontend y backend, donde la implementación era algo más complicada, pero la escalabilidad y la legibilidad eran mucho mejores, y abrían el panorama hacia un mundo de compiladores rápidos y sencillos de usar.

La evolución del compilador ha sido extraordinaria. Desde lenguaje binario para programar grandes máquina, hasta los intentos de lograr crear lenguajes más sencillos de leer, hasta crear código intermedio de manera manual, para luego investigar y desarrollar soluciones automatizadas a éstos dilemas, y, después de muchos intentos de optimización, el compilador ha llegado a lo que es hoy.

El compilador nos permite dominar el mundo de la tecnología, pues permite al programador escribir cientos de líneas de código, con la seguridad de que el compilador nos dirá dónde nos estamos equivocando, simplificando así nuestros esfuerzos por depurar cientos de líneas de código, y lograr que nuestros programas compilen en cualquier máquina, eliminando así las barreras del pasado, y permitiendo que nuestras soluciones lleguen a más personas, en más partes del mundo.

El humano siempre se ha caracterizado por ser una especie que busca las mejoras en su tecnología, y la investigación y nacimiento del compilador no es excepción.

# Bibliografía

*Historia de la Construcción de los Compiladores*. (25 de Noviembre de 2015). Obtenido de https://es.wikipedia.org/wiki/Historia\_de\_la\_construcci%C3%B3n\_de\_los\_compiladores

Knuth, D. E. (1971). *Springer.* Obtenido de http://link.springer.com/article/10.1007%2FBF00289517#page-1

Llatas, C. F. (s.f.). Estructura de Un Compilador.

*Modern Programming Timeline*. (22 de Enero de 2016). Obtenido de whoishostingthis.com: http://www.whoishostingthis.com/resources/programming/

*QUORA: What are advantages of frontend and backend in compilers?* (30 de October de 2015). Obtenido de QUORA: https://www.quora.com/What-is-advantages-of-frontend-and-backend-in-compiler

Serna-Perez, E. (2016). *Fases de Compilación*. Obtenido de paginasprodiy.com: http://www.paginasprodigy.com/edserna/cursos/compilador/notas/Notas1.pdf

Serna-Pérez, P. E. (2016). *Introducción a Compiladores.* Obtenido de http://www.paginasprodigy.com/edserna/cursos/compilador/notas/Notas1.pdf

*Timeline of Computer History*. (2016). Obtenido de computerhistory.org: http://www.computerhistory.org/timeline/1960/