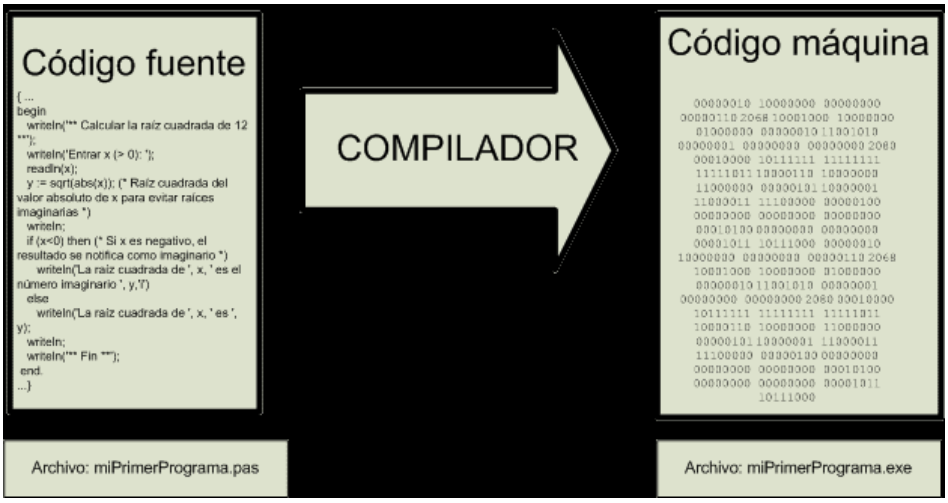
Nombre del alumno: __Josué Barrales Gálvez_____Matricula: _20191390_____

Nota: En el tema de desarrollo de Compiladores e Intérpretes conteste las siguientes preguntas:

1 ¿Qué es un compilador?

Un compilador es un tipo de traductor que transforma un programa entero de un lenguaje de programación a otro. Usualmente el lenguaje objetivo es código máquina, aunque también puede ser traducido a un código intermedio o a texto.



2 ¿Cuáles son las partes de un compilador?

Un compilador está dividido en 2 partes principales que son el frontend y el backend.

El frontend es la primera parte, por así decir la puerta de entrada para procesar el código fuente. La teoría que hay detrás de esta parte se puede emplear en muchas otros campos de la programación y es esencial Las tres principales partes de este frontend son:

1. Analizador léxico o scanner. Esta parte tiene como objetivo convertir la entrada en tokens, eliminar espacios en blanco, comentarios, etc. Estos tokens se comparten con el analizador sintáctico a través de un buffer, que puede ser de distintos tipos pero que en este caso no vamos a explicar con más detalles.
2. El analizador sintáctico o parser es la parte del compilador que tiene como finalidad construir una estructura de árbol con los tokens que recibe del analizador léxico. Este árbol es el AST (Abstract Syntax Tree).
3. El analizador semántico, que tiene como finalidad la realización de aquellas comprobaciones que el analizador sintáctico no puede hacer. Algunas de estas comprobaciones son los tipos de datos, que las constantes no tengan nuevas asignaciones. En resumen, tiene como fin obtener un árbol de derivación o AST verificado.

4. Generador de código intermedio. Esta parte genera un código intermedio que puede ser en ensamblador o similar. Se crea mediante instrucciones y con estructuras de datos como listas. Después de esta fase se puede en un fichero la salida del código como una cadena de texto formateada para comprobar si el código intermedio se ha generado correctamente. Este código intermedio se puede usar para las últimas dos fases del backend.

3 ¡Menciones los tipos de compiladores!

1. Compiladores cruzados: generan código para un sistema distinto del que están funcionando.
2. Compiladores optimizadores: realizan cambios en el código para mejorar su eficiencia, pero manteniendo la funcionalidad del programa original.
3. Compiladores de una sola pasada: generan el código máquina a partir de una única lectura del código fuente.
4. Compiladores de varias pasadas: necesitan leer el código fuente varias veces antes de poder producir el código máquina.
5. Compiladores JIT (just intime): forman parte de un intérprete y compilan partes del código según se necesitan.

4 ¡Escriba los elementos a que se deben tomar en cuenta para desarrollar un compilador!

1. Análisis: se trata de la comprobación de la corrección del programa fuente, según la definición del lenguaje en términos de teoría de lenguajes formales. Incluye las fases correspondientes al análisis léxico (que consiste en la descomposición del programa fuente en componentes léxicos), análisis sintáctico (agrupación de los componentes léxicos en frases gramaticales) y análisis semántico (comprobación de la validez semántica de las sentencias aceptadas en la fase de análisis sintáctico).
2. Síntesis: su objetivo es la generación de la salida expresada en el lenguaje objeto y suele estar formado por una o varias combinaciones de fases de generación de código (normalmente se trata de código intermedio o de código objeto) y de optimización de código (en las que se busca obtener un programa objetivo lo más eficiente posible, según su complejidad computacional o complejidad de Kolmogórov: tiempo de ejecución, espacio durante ejecución, espacio para ser almacenado fuera de ejecución, etc).
3. Alternativamente, las fases descritas para las tareas de análisis y síntesis se pueden agrupar en:
4. Analizador o front-end: es la parte que analiza el código fuente, comprueba su validez, genera el árbol de derivación y rellena los valores de la tabla de símbolos. Esta parte suele ser independiente de la plataforma o sistema para el cual se vaya a compilar, y está compuesta por las fases comprendidas entre el análisis léxico y la generación de código intermedio.
5. Generador o back-end: es la parte que genera el código máquina, específico de una plataforma, a partir de los resultados de la fase de análisis, realizada por este generador.

5 ¿Cómo se clasifican de los compiladores?

1. Una sola pasada: se examina el código fuente una vez, generando el código o programa objeto.
2. Pasadas múltiples: se requieren pasos intermedios para producir un código en otro lenguaje, y una pasada final para producir y optimizar el código producido durante los pasos anteriores.
3. Optimación: lee un código fuente, lo analiza y descubre errores potenciales sin ejecutar el programa.
4. Incrementales: generan un código objeto instrucción por instrucción cuando el usuario teclea cada orden individual.
5. Ensamblador: el lenguaje fuente es el lenguaje ensamblador (assembler) y posee estructura sencilla.
6. Compilador cruzado: genera código en lenguaje objeto (binario) para una máquina diferente de la que se está usando para compilar. Un ejemplo es construir un compilador de Pascal que genere código para MS-DOS y que el compilador funcione en LINUX y se haya escrito en C++.
7. Compilador con montador: compilador que compila distintos módulos de forma independiente y después es capaz de enlazarlos.
8. Autocompilador: compilador que está escrito en el mismo lenguaje que va a compilar. Evidentemente, no se puede ejecutar la primera vez. Sirve para hacer ampliaciones al lenguaje, mejorar el código generado, etc.
9. Metacompilador: se refiere a un programa que recibe como entrada las especificaciones del lenguaje para el que se desea obtener un compilador y genera como salida el compilador para ese lenguaje. El desarrollo de los metacompiladores se encuentra con la dificultad de unir la generación de código con la parte de análisis. Lo que sí se han desarrollado son generadores de analizadores léxicos y sintácticos
10. Descompilador: es un programa que acepta como entrada código máquina y lo traduce a un lenguaje de alto nivel, realizando el proceso inverso a la compilación.

6 ¿Por qué se debe desarrollar compiladores?

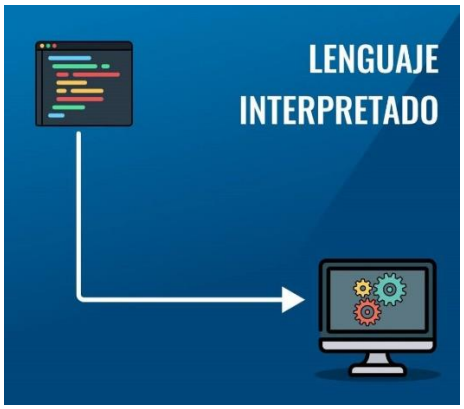
La importancia de los compiladores radica en que, sin estos programas no existiría ninguna aplicación informática, ya que son la base de la programación en cualquier plataforma.

Así que, por tanto, tenemos la paradoja de que para hacer un programa, necesitamos un compilador, que a su vez es un programa informático.

Además de que la función principal que cumple un compilador es traducir a un lenguaje mucho más sencillo y entendible por la máquina.

7 ¿Que es un Intérprete?

Un intérprete es un programa que analiza y ejecuta un código fuente, toma un código, lo traduce y a continuación lo ejecuta; y así sucesivamente lo hace hasta llegar a la última instrucción del programa, siempre y cuando no se produzca un error en el proceso.



8 ¿Mencione las partes de un Intérprete?

1. Traductor a Representación Interna: Toma como entrada el código del programa P en Lenguaje Fuente, lo analiza y lo transforma a la representación interna correspondiente a dicho programa P.
2. Representación Interna (P/RI): La representación interna debe ser consistente con el programa original. Entre los tipos de representación interna, los árboles sintácticos son los más utilizados y, si las características del lenguaje lo permiten, pueden utilizarse estructuras de pila para una mayor eficiencia.
3. Tabla de símbolos: Durante el proceso de traducción, es conveniente ir creando una tabla con información relativa a los símbolos que aparecen. La información a almacenar en dicha tabla de símbolos depende de la complejidad del lenguaje fuente. Se pueden almacenar etiquetas para instrucciones de salto, información sobre identificadores (nombre, tipo, línea en la que aparecen, etc.) o cualquier otro tipo de información que se necesite en la etapa de evaluación.
4. Evaluador de Representación Interna: A partir de la Representación Interna anterior y de los datos de entrada, se llevan a cabo las acciones indicadas para obtener los resultados. Durante el proceso de evaluación es necesario contemplar la aparición de errores.
5. Tratamiento de errores: Durante el proceso de evaluación pueden aparecer diversos errores como desbordamiento de la pila, divisiones por cero, etc. que el intérprete debe contemplar.

9 ¿Cuándo de sebe desarrollar un Intérprete?

Cuando necesitemos ejecutar códigos no muy grandes, o cuando necesitemos hacer pruebas y cambiar grandes segmentos del código,

10 ¿Cuáles son los tipos de Intérpretes?

1. Intérpretes Puros… Los intérpretes puros son los que analizan una sentencia y la ejecutan, y así sucesivamente todo el programa fuente. Fueron los intérpretes desarrollados en la primera generación de ordenadores, pues permitían la ejecución de largos programas con ordenadores de memoria muy reducida, ya que sólo debían contener en memoria el intérprete y la sentencia a analizar y ejecutar. El principal problema de este tipo de intérpretes es que si a mitad del programa fuente se producen errores, se debe de reiniciar el proceso.

2. Interpretes Avanzados… Los intérpretes avanzados o normales incorporan un paso previo de análisis de todo el programa fuente. Generando posteriormente un lenguaje intermedio que es ejecutado por ellos mismos. De esta forma en caso de errores sintácticos no pasan de la fase de análisis.
3. Interpretes Incrementales… Algunos lenguajes no se pueden compilar, debido a que entre sus características pueden manejar objetos o funciones que no son conocidos en tiempo de compilación, ya que son creados en ejecución. Para este tipo de lenguajes existen los intérpretes incrementales, que permiten compilar los módulos completamente definidos, y recompilar en tiempo de ejecución los nuevos módulos.

11 ¿Elementos que se deben tomar para el desarrollo de un Intérprete?

1. Análisis léxico
2. Análisis sintáctico
3. Análisis Semántico

12 ¿Cuáles son las diferencias entre Compiladores e Intérpretes?(haga una tabla).

Compilador	Intérprete
Tiene como salida un único lenguaje objeto.	Tiene como salida instrucciones traducidas
El rendimiento en la ejecución del programa compilado (la salida) es más rápido que interpretado.	El rendimiento (del intérprete) se somete al rendimiento de analizar la traducción una a una.
La salida puede depender de la arquitectura.	Tiende a ser más portable e independiente de la arquitectura.
No requiere del programa fuente porque el programa objeto es ejecutable. **Puede ser secreto**(El programa fuente)	Se requiere del lenguaje fuente para su ejecución.
Los errores sintácticos y semánticos se detectan antes de la ejecución del programa objeto.	Se detectan los errores en la ejecución del programa.
Tiene menos flexibilidad en el uso de la memoria para el programa objeto.	Es más flexible para que el programa pueda usar la memoria.