Nombre: Brayan Jose Yañez Betancur

Correo: [brayanbetancur2619@gmail.com](mailto:brayanbetancur2619@gmail.com)

Grupo: T2

**INVESTIGACION**

**Compiladores e interpretes**

Un compilador es un **programa informático** que traduce todo el código fuente de un proyecto de software a código máquina antes de ejecutarlo. Solo entonces el procesador ejecuta el software, obteniendo todas las instrucciones en código máquina antes de comenzar. De esta manera, el procesador cuenta con todos los componentes necesarios para ejecutar el software, procesar las entradas y generar los resultados. No obstante, en muchos casos, durante el proceso de compilación tiene lugar un paso intermedio fundamental: antes de generar la traducción final en código máquina, la mayoría de los compiladores suelen convertir el código fuente en un **código intermedio** (también llamado código objeto) que, a menudo, es compatible con diversas plataformas y que, además, también puede ser utilizado por un intérprete.

Al producir el código, el compilador determina **qué instrucciones** van a enviarse al procesador y **en qué orden.** Si las instrucciones no son interdependientes, incluso es posible que puedan procesarse en paralelo.

Un intérprete es un **programa informático** que procesa el código fuente de un proyecto de software durante su tiempo de ejecución, es decir, mientras el software se está ejecutando, y actúa como una **interfaz** entre ese proyecto y el procesador. Un intérprete siempre procesa el código línea por línea, de modo que lee, analiza y prepara cada secuencia de forma consecutiva para el procesador. Este principio también se aplica a las secuencias recurrentes, que se ejecutan de nuevo cada vez que vuelven a aparecer en el código. Para procesar el código fuente del software, el intérprete recurre a sus propias bibliotecas internas: en cuanto una **línea de código fuente se ha traducido a los correspondientes comandos legibles por máquina**, esta se envía directamente al procesador.

El proceso de conversión no finaliza hasta que se ha interpretado todo el código. Solo se interrumpe prematuramente si se produce un fallo durante el procesamiento, lo que **simplifica mucho la resolución de los errores**, ya que la línea de código problemática se detecta inmediatamente después de ocurrir el fallo.

**Lenguajes de tipado fuerte y tipado débil**

Tipado fuerte Es el utilizado para que el código incluya el tipo de dato al declarar la variable. Solemos decir que esta forma de programar es más expresiva al saber qué sucederá exactamente con las órdenes programadas. La ejecución es más veloz, ya que la inferencia de los tipos es anterior y el lenguaje no necesita verificarlos a la hora de ejecutarlos. Se usa en lenguajes de programación como Python, C, C#, Java y Ruby. Sus desventajas son:

* A la hora de programar es mucho más estricto y necesitas escribir más código, lo que alarga el tiempo de trabajo y aumenta la necesidad de verificar lo redactado más exhaustivamente.
* El compilador debe ejecutar una disciplina de tipos concreta. Es decir, se siguen unas normas concretas de asignación de variables, argumentos de funciones o valores de retorno.

Tipado débil Se usa para indicar el tipo de variable a la hora de declararla. Permite asignarle un valor entero a una variable que con anterioridad tenía una cadena. En programación informática se usa en lenguajes como el JavaScript o PHP. Presenta desventajas como:

* Es muy rápido de desarrollar, pero favorece la aparición de errores.
* Su flexibilidad ayuda a que los programas funcionen sin necesidad de utilizar tanto código. No obstante, presenta errores en el tiempo de ejecución, como el que se pierde al comparar números mezclando algunos tipados como los strings, que no son detectables anteriormente.
* Es necesario castear de manera periódica el programa para conseguir el resultado esperado. Ello provoca que tengas que redactar más código para aumentar su seguridad.

**Diferencias entre ambos tipados**

El tipado fuerte admite realizar operaciones con distintos tipos de variable. La consecuencia directa es que no vas a cometer tantos errores como con el lenguaje de tipado débil. Ahora bien, necesitas escribir más código. Este lenguaje es ideal para proyectos de mayor alcance y con un número mayor de especificaciones.

El tipado débil te permite trabajar en menos tiempo y comprobar cuáles son los resultados del programa en tiempo real. Para la fase de ideación del proyecto esta opción es ideal. Se adapta muy bien a programas de menor capacidad.