MEDAC
Instituto Oficial de Formación Profesional

ENTORNOS DE DESARROLLO

Explotación de código

3

ÍNDICE

OBJETIVOS

Conocer los tipos de lenguajes de programación.

Conocer las diferentes herramientas que intervienen en la creación de soluciones software.

Entender los principios de la reutilización de código.

Conocer el concepto de depuración software.

/ 1. Introducción y contextualización práctica

El desarrollo software requiere de un conjunto de herramientas imprescindibles para la generación de programas informáticos. Para poder crear soluciones que se adapten mejor a los problemas, es importante conocer todas las posibilidades existentes a la hora de elegir un lenguaje u otro.

Una vez que seleccionamos un lenguaje de programación, es necesario escribir la solución de nuestro problema con dicho lenguaje y las posibles dependencias externas que necesitemos utilizar. Para poder probar nuestra solución es necesario generar un código que pueda entender el computador, a través de un proceso de traducción al lenguaje que la máquina pueda entender. Cuando nuestro código se ejecuta, pueden producirse errores lógicos que solo son detectables durante la ejecución.

Como desarrolladores debemos saber cómo solucionarlos. Sin embargo, en programas grandes suele ser más complicado encontrar los errores, que realizar su corrección. Para facilitar la búsqueda de errores en el software, podemos utilizar depuradores.

Fig. 1. Es necesario depurar el código para que no tenga errores

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Audio intro. ". Lenguaje ensamblador e interpretación "

https://bit.ly/2PHvivT

/ 2. Proceso de obtención de código

La creación de un programa informático requiere de un determinado conjunto de herramientas que permitan decir a la máquina qué realizar.

Actualmente, los lenguajes de programación se parecen a lenguaje humano, por ello son considerados lenguajes de alto nivel. Sin embargo, un computador es una máquina con una electrónica que solo entiende impulsos eléctricos dentro de determinado voltaje. En este contexto cabe preguntarse cómo es posible convertir un programa escrito en lenguaje pseudonatural a impulsos eléctricos.

Para que un computador pueda ejecutar un código fuente es necesario que éste sea entendido por la CPU o procesador. Cada procesador es capaz de entender un conjunto determinado de instrucciones en un determinado lenguaje, que recibe el nombre de lenguaje ensamblador o lenguaje máquina. De esta forma el lenguaje ensamblador es el más cercano a la máquina, siendo el nivel más bajo de abstracción en el que se puede crear un software.

Como desarrolladores de software en muy pocas ocasiones tendremos que lidiar con este tipo de lenguaje, pues el código que se genera va a depender del procesador en el que se va a ejecutar.

Mediante la creación de lenguajes de alto nivel de abstracción, somos capaces de crear programas informáticos independientes del procesador con mayor facilidad. Sin embargo, un procesador solo es capaz de entender lenguaje ensamblador, por ello es necesario crear herramientas que permitan traducir lenguajes de alto nivel a lenguaje ensamblador.

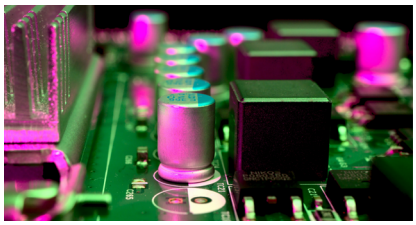

Fig. 2. Placa que puede ser programada en ensamblador

Video 1. "Generación de analizadores léxicos "

https://bit.ly/33O7rCZ

/ 3. Características de los lenguajes de programación

Los lenguajes de programación de alto nivel se parecen a un inglés con ciertas particularidades. Independientemente del tipo de lenguaje de programación, para poder crear un programa informático es necesario disponer de las herramientas necesarias que permita escribir el código fuente del programa.

Normalmente, se utilizan herramientas muy potentes, pero podría ser suficiente con editores de texto plano. Además, es necesario que existan herramientas que permitan realizar la traducción del lenguaje a un lenguaje ensamblador. En este grupo herramientas tendríamos que incluir los compiladores, los intérpretes, los enlazadores, etc.

Además de las herramientas de desarrollo, es necesario conocer ciertas características de cada uno de los lenguajes de programación, pues es lo que diferencia un lenguaje de otro. Desde el punto de vista del programador, un lenguaje de programación está formado por un léxico, la sintaxis y unas reglas semánticas:

- El **léxico** nos indica qué caracteres puede reconocer el lenguaje de programación.
- La **sintaxis** nos indica cuál es el orden correcto en el que deben aparecer los determinados elementos del lenguaje.
- Las **reglas semánticas** nos dicen qué operaciones desde el punto de vista lógico se permiten en el lenguaje.

Lenguajes compilados

Los lenguajes compilados son aquellos que requieren de un compilador para poder generar el código ensamblador.

Un compilador es una herramienta que permite traducir de un lenguaje de alto nivel a un lenguaje que pueda comprender la máquina. Este tipo de lenguajes diferencian la etapa de compilación de la etapa de ejecución. Como podemos deducir, al ahorrarnos la etapa de compilación continuamente tendremos un mejor rendimiento.

Para poder realizar la compilación y obtener un programa ejecutable, es necesario disponer también de un enlazador o linker. Esta herramienta es la que permitirá resolver las dependencias que tengamos en nuestro desarrollo, por ejemplo, con la resolución de dependencias con componentes externos.

Audio 1. "Herramientas para la construcción de compiladores"
https://bit.ly/33KHJiR

/ 4. Caso práctico 1: "Sistema para control de navegación"

Planteamiento. Nuestro equipo desea crear un sistema de control de navegación para barcos utilizando un lenguaje ensamblador genérico, que según nos dicen desde el departamento de plataformas, es lo más recomendable y más éxito nos va a traer, ya que es compatible con todas las CPUs del mercado.

Nudo. ¿Crees que sería un buen enfoque?

Desenlace. Según hemos visto, el lenguaje ensamblador es lo más cercano a la CPU y dependerá de cada una de las arquitecturas que posea.

La creación de una solución software mediante la utilización de código ensamblador sería muy costosa y poco realista.

Por un lado, sería muy costosa porque requeriría mucho tiempo de desarrollo ya que suele ser un lenguaje complicado y que requiere muchas instrucciones para realizar poco (escribe mucho realiza poco).

Por otro lado, sería una solución poco realista porque a poco que cambiase un componente hardware en el barco, habría que realizar una comprobación de que todo el software sigue funcionando.

Ante esta tesitura cabría preguntarse, ¿qué sucedería si se rompe un componente y no encontramos repuesto para ese tipo de arquitectura?... en tal caso habría que adquirir otro componente para el cual, nuestro código no estaría preparado...

En este caso, lo mejor sería utilizar un lenguaje de alto nivel que permita adatarse a cada uno de los módulos del barco. Sería posible utilizar el código ensamblador para realizar pequeñas tareas de ajuste por temas de rendimiento, pero no sería una buena idea construir toda la solución en código máquina.

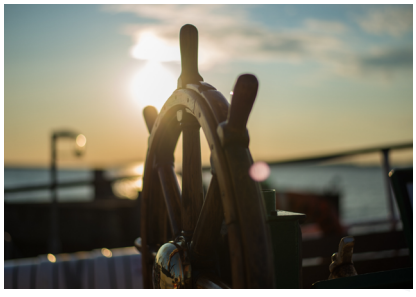

Fig. 3. Sistema de navegación de un barco

/ 5. Lenguajes interpretados, errores y depuradores

a. **Lenguajes interpretados**

Los lenguajes interpretados son aquellos que no diferencian entre la fase de compilación y la fase de ejecución.

Este tipo de lenguajes requieren que cada vez que se vaya a ejecutar el programa, se realice una compilación previa. Suelen estar muy vinculados con campos como la inteligencia artificial en los que se requiere un tipo de programación diferente. Sin embargo, su utilización no es exclusiva a este campo, pues existen lenguajes de propósito general que pueden ser interpretados en alguna de sus fases, por ejemplo, Java o .NET.

Java es un lenguaje que mezcla la compilación con la interpretación. La compilación se realiza desde un lenguaje de alto nivel a un lenguaje ensamblador, sin embargo, Java realiza una traducción a un lenguaje intermedio que recibe el nombre de bytecode.

b. **Errores y depuradores**

Una vez que hemos escrito la solución a nuestro problema es necesario generar un ejecutable para poder realizar las pruebas necesarias. Muchas veces los desarrolladores se dan por satisfechos cuando el programa compila, es decir, se ha generado un ejecutable, sin embargo, en este punto solo hemos probado que sintácticamente el programa es capaz de compilar, es decir, tener su traducción al código ensamblador.

Como podemos deducir, que un programa compile no quiere decir que su funcionamiento sea el correcto, pues puede haber fallos lógicos que solo aparecerán cuando el programa esté en ejecución. Por este motivo, es necesario realizar una etapa de pruebas y depuración.

Los depuradores son herramientas que nos permiten ver el contenido de las variables y el flujo de ejecución del programa en tiempo real. Además, los depuradores nos permiten parar el flujo de ejecución del programa en cualquier momento pudiendo ir hacia delante con más o menos rapidez. Para poder utilizar el depurador de un programa que hayamos creado es necesario disponer de las herramientas de desarrollo que estimemos oportunas. Los depuradores son, por tanto, herramientas que nos permiten solventar errores lógicos durante la ejecución de un programa.

En el siguiente vídeo podrás ver cómo es la depuración de un programa usando el IDE Netbeans (concepto que estudiaremos en la siguiente unidad y que profundizarás en la asignatura de Programación)

Video 2. "Depuración de programas"
https://bit.ly/2Zxx89k

/ 6. Reutilización de código

Un buen desarrollo software se centra en crear código estructurado e independiente de un contexto determinado. Si somos capaces de crear componentes que no dependan de un problema determinado, será posible poder reutilizar ese componente o ese código en cualquier contexto en el futuro.

Existen diferentes formas de garantizar la reutilización de software, por ejemplo: mediante funciones y/o procedimientos, con la creación de componentes, etc.

La reutilización del código es un campo que estudia la ingeniería del software, teniendo como objetivo ahorrar tiempo y recursos mediante la eliminación de redundancias en el mismo. La idea principal es poder reutilizar partes de un programa que ya han sido escritas en otros contextos para poder agilizar el desarrollo de nuevos programas.

Como se puede intuir, la reutilización de código puede requerir la creación de componentes reutilizables e independientes que pueden ser de cualquier tipo, por ejemplo, pruebas genéricas, entornos de depuración, etc.

Una buena aproximación a la reutilización de código es la creación de librerías que se pueden publicar y ser utilizadas por terceras partes. Al tener librerías que realizan funcionalidades muy acotadas, es más sencillo crear pruebas simples que sirvan para detectar errores.

Un ejemplo de la utilización de estas librerías es Maven o CocoaPods para el desarrollo principalmente sobre iOS.

Fig. 4. Recicla tu código para ahorrar tiempo y recursos

/ 7. Caso práctico 2: "Control del nivel de agua"

Planteamiento. Nuestro jefe nos encarga un proyecto para el control de los sensores de las compuertas de una presa. Los sensores instalados en diferentes partes de la instalación enviarán en tiempo real, el nivel del agua al receptor más cercano. Nuestro software deberá analizar dichos valores y tomar la decisión en tiempo real de si es necesario abrir o no las compuertas.

Desde la dirección de la empresa, nos dicen que les gustaría crear dicha solución mediante lenguajes interpretados dada su versatilidad.

Nudo. ¿Crees que sería un buen enfoque la solución que nos proponen?

Desenlace. Según hemos visto a lo largo del tema, los lenguajes interpretados pueden introducir cierta latencia a la hora de tomar decisiones en tiempo real. Esta latencia puede ser asumible en determinados contextos, sin embargo, en aquellos en los que hay vidas humanas en peligro, lo recomendable es utilizar otro tipo de soluciones.

Esto mismo se puede aplicar a soluciones que controlan el sistema de navegación de un avión o los sistemas de control de un túnel. Por ello, sería mejor utilizar una solución compilada y, si es posible, con lenguajes más sencillos como C.

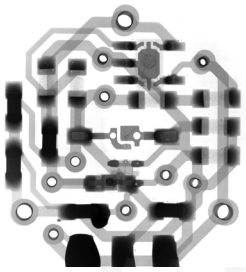

Fig. 5. Utilización de sensores para obtener datos del entorno

/ 8. Resumen y resolución del caso práctico de la unidad

En este tema hemos presentado las principales características de los lenguajes de programación desde el punto de vista de la generación del código que ejecuta la CPU/procesador.

Como hemos visto, los lenguajes pueden ser compilados o interpretados, requiriendo diferentes tipos de herramientas para ello.

El proceso de compilación y/o interpretación es necesario porque los lenguajes actuales no son comprendidos directamente por el procesador y requieren de un proceso de traducción. En la creación de una solución informática es necesario tener un punto de vista general e independiente del contexto que nos permita generar soluciones que no dependan de la máquina ni el problema en cuestión.

Normalmente, en la primera ejecución de un programa suelen haber errores lógicos que pueden analizarse mediante la utilización de herramientas de depuración. Nótese que, como norma general, el tiempo de búsqueda de los errores suele ser mucho mayor que el tiempo requerido para la solución de éste.

Resolución del caso práctico de la unidad

Como hemos podido ir deduciendo a lo largo de la unidad, cada tipo de lenguaje de programación tiene un paradigma en el que se puede utilizar. Así, para el desarrollo en entornos web es mejor utilizar lenguajes que estén relacionados con HTML, CSS, etc.

Al pedirnos que desarrollemos una solución web en un entorno ensamblador y unirlo con lenguaje interpretados no tiene mucho sentido.

Por un lado, el lenguaje ensamblador no es un lenguaje apto para el desarrollo web. Por otro lado, los lenguajes interpretados pueden realizar algún tipo de procesamiento, pero no utilizarse en la maquetación web.

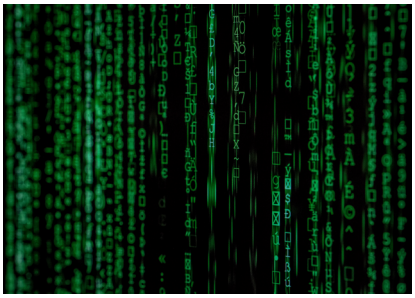

Fig. 6. Aprende a programar programando y depurando

/ 9. Bibliografía

Fowler, M. & Beck, K. (2019). Refactoring: improving the design of existing code. Boston: Addison-Wesley.

AGANS, D. (2013). DEBUGGING: the 9 indispensable rules for finding even the most elusive software and hardware ... problems. Place of publication not identified: AMACOM.

Aho, A., Sethi, R. & Ullman, J. (1986). Compilers, principles, techniques, and tools. Reading, Mass: Addison-Wesley Pub. Co.