Preparador Informática

www.preparadorinformatica.com

TEMA 29 INFORMÁTICA / TEMA 31 S.A.I.

UTILIDADES PARA EL DESARROLLO Y PRUEBAS DE PROGRAMAS. COMPILADORES. INTÉRPRETES. DEPURADORES.

TEMA 29 INF/ TEMA 31 SAI: UTILIDADES PARA EL DESARROLLO Y PRUEBAS DE PROGRAMAS. COMPILADORES. INTÉRPRETES. DEPURADORES.

1. INTRODUCCIÓN

Los lenguajes de programación están específicamente diseñados para programar computadores. Entre sus características se encuentran que son independientes de la arquitectura física del computador, y que utilizan notaciones cercanas a las habituales en el ámbito en que se usan de forma que las operaciones se expresan con sentencias o frases muy parecidas al lenguaje matemático o al lenguaje natural.

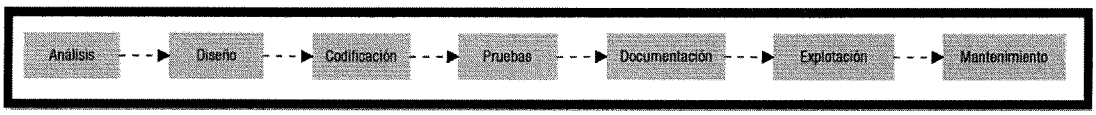
Como consecuencia de este alejamiento de la máquina y acercamiento a las personas, los programas escritos en lenguajes de programación no pueden ser directamente interpretados por un computador, siendo necesario realizar previamente su traducción a lenguaje máquina.

El presente tema está dedicado a estudiar las utilidades para el desarrollo y prueba de programas, como son los entornos de desarrollo integrados (IDE) y herramientas para realizar pruebas de programas. En el tema se presta especial atención a varios de los componentes que suele integrar un IDE, como son los compiladores, intérpretes y depuradores.

2. UTILIDADES PARA EL DESARROLLO Y PRUEBAS DE PROGRAMAS

Se entiende por desarrollo de programas a todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.

El desarrollo de software es un proceso que conlleva una serie de pasos. Genéricamente, estos pasos son los siguientes:



Centrándonos en la codificación y pruebas, dichas tareas se llevan a cabo por medio de entornos de desarrollo integrados y de frameworks.

A continuación, se detallan las funciones y características de dichas utilidades.

2.1. ENTORNOS DE DESARROLLO

Los entornos de desarrollo integrados (IDE) son una combinación de herramientas que permiten automatizar el proceso de desarrollo y prueba de los programas.

Algunas de sus características son:

- Facilitan las labores de programación.
- Aportan herramientas automáticas de ayuda.
- Son independientes respecto del entorno final de ejecución de la aplicación.
- Facilitan la compatibilidad y la integración en familias de productos.

2.1.1. FUNCIONES

Las principales funciones de los IDE son:

- Editor de código: coloración de la sintaxis.
- Auto-completado de código, atributos y métodos de clases.
- Identificación automática de código.
- Herramientas de concepción visual para crear y manipular componentes visuales.
- Asistentes y utilidades de gestión y generación de código.
- Compilación de proyectos complejos en un solo paso.
- Control de versiones.
- Generador de documentación integrado.
- Detección de errores de sintaxis en tiempo real.
- Refactorización de código: cambios menores en el código que facilitan su legibilidad sin alterar su funcionalidad (por ejemplo, cambiar el nombre a una variable).

- Permite introducir automáticamente tabulaciones y espaciados para aumentar la legibilidad.
- Depuración: seguimiento de variables, puntos de ruptura y mensajes de error del intérprete.
- Aumento de funcionalidades a través de la gestión de sus módulos y plugins.
- Administración de las interfaces de usuario (menús y barras de herramientas).
- Administración de las configuraciones del usuario.

2.1.2. COMPONENTES

Los entornos de desarrollo, ya sean libres o propietarios, están formados por una serie de componentes software que determinan sus funciones.

- Editor de textos
- Compilador/intérprete
- Depurador
- Interfaz gráfica

En los siguientes apartados se desarrollarán en profundidad los conceptos de compilador, intérprete y depurador.

2.1.3. ENTORNOS INTEGRADOS LIBRES Y PROPIETARIOS

Los entornos integrados de desarrollo más relevantes en la actualidad son:

a) Entornos Integrados Libres

Son aquellos con licencia de uso público.

Tipos de entornos de desarrollo libres más relevantes en la actualidad.		
IDE	**Lenguajes que soporta**	**Sistema Operativo**
NetBeans.	C/C++, Java, JavaScript, PHP, Python.	Windows, Linux, Mac OS X.
Eclipse.	Ada, C/C++, Java, JavaScript, PHP.	Windows, Linux, Mac OS X.
Gambas.	Basic.	Linux.
Anjuta.	C/C++, Python, Javascript.	Linux.
Geany.	C/C++, Java.	Windows, Linux, Mac OS X.
GNAT Studio.	Fortran.	Windows, Linux, Mac OS X.

b) Entornos Integrados Propietarios

Son aquellos entornos integrados de desarrollo que necesitan licencia.

Tipos de entornos de desarrollo propietarios más relevantes en la actualidad.		
IDE	**Lenguajes que soporta**	**Sistema Operativo**
Microsoft Visual Studio.	Basic, C/C++, C#.	Windows.
FlashBuilder.	ActionScript.	Windows, Mac OS X.
C++ Builder.	C/C++.	Windows.
Turbo C++ profesional.	C/C++.	Windows.
JBuilder.	Java.	Windows, Linux, Mac OS X.
JCreator.	Java.	Windows.
Xcode.	C/C++, Java.	Mac OS X.

2.2. HERRAMIENTAS PARA LAS PRUEBAS

Entre las principales herramientas que nos podemos encontrar en el mercado, para poder realizar las pruebas, las más destacadas son:

Tipo de prueba	Herramienta	Lenguaje
UNITARIA	JUnit	Java
	Simple Test	PHP
	PHP Unit	PHP
	Jtiger	Java
FUNCIONAL Y DE REGRESIÓN	Selenium	Java, PHP, Python, Ruby, etc.
	Watir	Ruby
	Watij	Java
	JWebUnit	Java
	NUnit	C#, J#, VB y C++
	Http Unit	Java
INTEGRACIÓN	TestNG	Java
	Hudson	Java
	Continuum	Java
CARGA Y RENDIMIENTO	JMeter	Java
	Jcrawler	Java
ACEPTACIÓN	FitNesse	Java, PHP, Ruby, .NET
	Concordion	Java, Python, Ruby, .NET
	Nessus	NASL,XML,HTML

3. TRADUCTORES

Debido a que los computadores únicamente pueden interpretar y ejecutar código máquina, existen traductores, que traducen programas escritos en lenguajes de programación a lenguaje máquina. Un traductor es, por tanto, un programa que recibe como entrada un código en un lenguaje de programación concreto, y produce, como salida, un código en lenguaje máquina equivalente.

El programa inicial se denomina programa fuente o **código fuente**, y el programa obtenido, programa objeto o **código objeto**.

El proceso de traducción de código fuente a código objeto puede realizarse de dos formas:

- Compilación: El proceso de traducción se realiza sobre todo el código fuente y el software responsable se llama **compilador**.
- Interpretación: El proceso de traducción del código fuente se realiza línea a línea y se ejecuta simultáneamente. No se genera código objeto intermedio. El software responsable se llama **intérprete**.

Existen lenguajes cuyos traductores se idearon como intérpretes (Basic, APL, Lisp, Prolog, Perl, Ruby, Javascript, etc.), y otros como compiladores (Fortran, Cobol, C, C++, etc). Un caso notable es el del lenguaje de programación Java, que consigue una gran portabilidad gracias a la descomposición del proceso de traducción en dos fases. En la primera el programa se traduce a un código intermedio normalizado (bytecode) y, por tanto, portable. En la segunda fase este código a través de la JVM (Java Virtual Machine) pasa a código máquina, ya directamente ejecutable por la computadora.

3.1. COMPILADORES

La traducción por un compilador, es decir el proceso de compilación, consta de dos etapas fundamentales: la etapa de análisis del programa fuente y la etapa de síntesis del programa objeto. Cada una de estas etapas conlleva la realización de varias fases. El análisis del texto fuente conlleva la realización de un análisis

del léxico, de la sintaxis y de la semántica. La síntesis del programa objeto conlleva la generación de código y su optimización.

ANÁLISIS

- **Análisis léxico**: se comprueba que los símbolos del lenguaje se han escrito correctamente y se agrupan en tokens.

- **Análisis sintáctico**: se comprueba que el resultado de la etapa anterior obedece a la gramática del lenguaje y aparecen en el orden correcto.

- **Análisis semántico**: se comprueba que el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. Durante la fase de análisis semántico se pueden producir errores, cuando se detectan construcciones sin un significado correcto. Por ejemplo, asignar a una variable definida como dato numérico en simple precisión el valor de una variable cadena de caracteres.

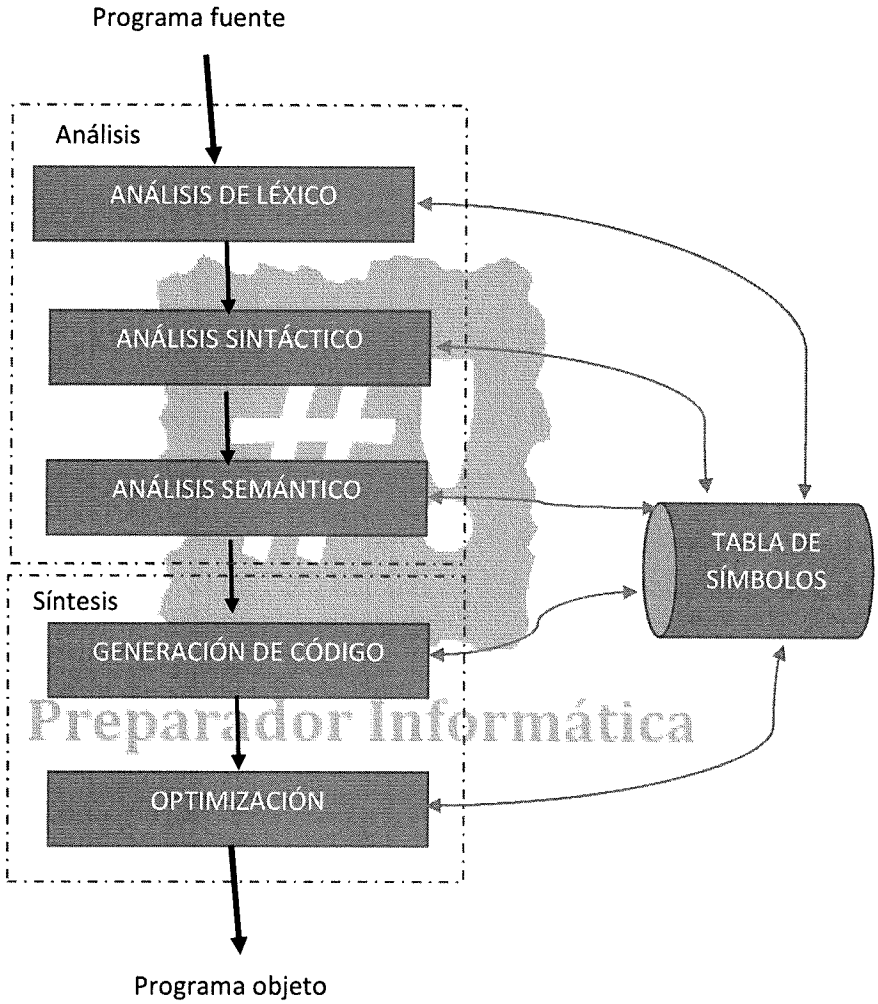
SÍNTESIS

- **Generación y optimización de código**: En esta fase se crea un archivo con un código en lenguaje objeto (normalmente lenguaje máquina) con el mismo significado que el texto fuente. El archivo objeto generado puede ser (dependiendo del compilador) directamente ejecutable, o necesitar otros pasos previos a la ejecución, tales como ensamblado, encadenado y carga.

  En la generación de código intermedio se completan y consultan las tablas generadas en fases anteriores (tablas de símbolos, de constantes, etc.). También se realiza la asignación de memoria a los datos definidos en el programa.

  En la fase de optimización se mejora el código intermedio, analizándose el programa objeto globalmente. Por ejemplo, un programa puede incluir dentro de un bucle que debe ejecutarse diez mil veces, una sentencia que asigna a una variable un valor constante (B=7.5), no alterándose dicho

valor (B) en el bucle. Con ello, innecesariamente se asignaría el valor 7.5 a la variable B diez mil veces. El optimizador sacaría la sentencia B=7.5 fuera (antes) del bucle, ejecutándose así dicha instrucción una Sola vez. El programa inicial es correcto, pero la optimización realizada por el compilador reduce el tiempo de ejecución.



3.2. INTÉRPRETES

Un intérprete hace que un programa fuente escrito en un lenguaje vaya sentencia a sentencia traduciéndose y ejecutándose directamente por el computador. El intérprete capta una sentencia fuente, la analiza e interpreta dando lugar a su ejecución inmediata, no creándose, por tanto, un archivo o programa objeto almacenable en memoria.

Los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa mientras se ejecuta, pero a cambio son más flexibles como entornos de programación y depuración, y permiten ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete, sino del propio intérprete.

4. DEPURADORES

Durante el proceso de desarrollo de software, se pueden producir dos tipos de errores: errores de compilación o errores lógicos. Ejemplos de errores de compilación son si al escribir una sentencia, se olvida escribir un ";", se hace referencia a una variable inexistente o se utiliza una sentencia incorrecta. El programa no puede compilarse hasta que el programador no corrija ese error.

El otro tipo de errores son lógicos, comúnmente llamados bugs, estos no evitan que el programa se pueda compilar con éxito, ya que no hay errores sintácticos, ni se utilizan variables no declaradas, etc. Sin embargo, los errores lógicos, pueden provocar que el programa devuelva resultados erróneos, que no sean los esperados o pueden provocar que el programa termine antes de tiempo o no termine nunca.

Para solucionar los errores lógicos, se utiliza la herramienta conocida como **depurador (debugger).** El depurador permite supervisar la ejecución de los programas, para localizar y eliminar los errores lógicos. Un programa debe compilarse con éxito para posteriormente poder utilizarlo en el depurador. El depurador nos permite analizar todo el programa, mientras éste se ejecuta. Permite suspender la ejecución de un programa, examinar y establecer los

valores de las variables, comprobar los valores devueltos por un determinado método, el resultado de una comparación lógica o relacional, etc.

De este modo, el depurador permite analizar el flujo de ejecución del código y el estado de los datos conforme van siendo manipulados por el programa.

Para poder depurar un programa, podemos ejecutar el programa de diferentes formas, de manera que en función del problema que se quiera solucionar, resulte más sencillo un método u otro.

Existen los siguientes tipos de ejecución:

- **Paso a paso por instrucción**: Algunas veces es necesario ejecutar un programa línea por línea, para buscar y corregir errores lógicos.
- **Paso a paso por procedimiento**: permite introducir los parámetros que queremos a un método o función de nuestro programa, pero en vez de ejecutar instrucción por instrucción ese método, nos devuelve su resultado. Es útil, cuando hemos comprobado que un procedimiento funciona correctamente, y no nos interese volver a depurarlo, sólo nos interesa el valor que devuelve.
- **Ejecución hasta una instrucción:** el depurador ejecuta el programa, y se detiene en la instrucción donde se encuentra el cursor, a partir de ese punto, podemos hacer una depuración paso a paso o por procedimiento.
- **Ejecución hasta el final del programa**: se ejecuta las instrucciones de un programa hasta el final, sin detenernos en las instrucciones intermedias.

5. CONCLUSIÓN

En el presente tema se ha presentado una visión global de los entornos de desarrollo integrados, describiendo sus principales funciones y componentes más importantes e indicando además los entornos libres y propietarios más relevantes en la actualidad.

Como hemos visto, los entornos de desarrollo integrados están formados por una serie de componentes software que determinan sus funciones (editores,

compiladores, intérpretes, depuradores, etc). Se ha prestado especial atención a algunos de estos componentes como son los compiladores, intérpretes y depuradores, detallando para cada uno de ellos las funciones que llevan a cabo.

6. BIBLIOGRAFÍA

- Aho, A.V.; Sethi, R.; Ullman, J.D.: **Compiladores. Principios, técnicas y herramientas**. Editorial Addison Wesley.
- Prieto, A. Lloris, A. y Torres, J.C. **Introducción a la informática**. Editorial McGraw-Hill
- Sanchís Llorca, F.J.; Galán Pascual, G.: **Compiladores. Teoría y Construcción**. Editorial Paraninfo.
- Alcalde, E. y García, M. **Metodología de la Programación**. Editorial McGraw-Hill
- http://atc.ugr.es/APrieto_videoclases Departamento de Arquitectura y Tecnología de los Computadores. Universidad de Granada. Prieto, A.