

¡DROGAS  
NO  
SON  
MORFÍN  
Y  
CÓCAINA  
SON  
DROGAS  
¡DROGAS  
NO  
SON  
MORFÍN  
Y  
CÓCAINA  
SON  
DROGAS

Los lenguajes son sistemas de comunicación. Un lenguaje de programación consiste en todos los símbolos, caracteres y reglas de uso que permiten a las personas "comunicarse" con las computadoras. Existen por lo menos varios cientos de lenguajes y dialectos de programación diferentes. Algunos se crean para una aplicación especial, mientras que otros son herramientas de uso general más flexibles que son apropiadas para muchos tipos de aplicaciones. En todo caso los lenguajes de programación deben tener instrucciones que pertenecen a las categorías ya familiares de entrada/salida, cálculo/manipulación de textos, lógica/comparación y almacenamiento/recuperación.

No obstante, aunque todos los lenguajes de programación tienen un conjunto de instrucciones que permiten realizar dichas operaciones, existe una marcada diferencia en los símbolos, caracteres y sintaxis de los lenguajes de máquina, lenguajes ensambladores y lenguajes de alto nivel.

## Lenguajes de alto nivel

Los primeros programas ensambladores producían sólo una instrucción en lenguaje de máquina por cada instrucción del programa fuente. Para agilizar la codificación, se desarrollaron programas ensambladores que podían producir una cantidad variable de instrucciones en lenguaje de máquina por cada instrucción del programa fuente. Dicho de otra manera, una sola macroinstrucción podía producir varias líneas de código en lenguaje de máquina.

El desarrollo de las técnicas mnemotécnicas y las macroinstrucciones condujo, a su vez, al desarrollo de lenguajes de alto nivel que a menudo están orientados hacia una clase determinada de problemas de proceso.

A diferencia de los programas de ensamble, los programas en lenguaje de alto nivel se pueden utilizar con diferentes marcas de computadoras sin tener que hacer modificaciones considerables. Esto permite reducir sustancialmente el costo de la reprogramación cuando se adquiere equipo nuevo. Otras ventajas de los lenguajes de alto nivel son:

- Son más fáciles de aprender que los lenguajes ensambladores.
- Se pueden escribir más rápidamente.
- Permiten mejor documentación.

A continuación se examinarán algunos de los principales lenguajes de alto nivel.

# C

Este es un lenguaje especializado en la programación de sistemas. Fue diseñado alrededor de 1970 por Dennis Ritchie, de los Laboratorios Bell. Se emplea para escribir compiladores y sistemas operativos; actualmente el lenguaje C es uno de los más usados junto con sus derivados (visual C, C+, C++, etc).

El lenguaje C produce un código parecido al escrito en ensamblador, con la ventaja de que éste es un lenguaje de alto nivel. Un ejemplo de esto es que se pueden asignar registros de la UCP durante la compilación y llamar a rutinas y subsistemas del sistema operativo desde el programa, sobre todo cuando se emplea desde Unix.

**Un programa en C consta de módulos que pueden llamarse recursivamente, pero no de manera anidada. El manejo de la memoria es dinámico. Lenguaje de programación C**

De Wikipedia, la enciclopedia libre.

C es un lenguaje de programación creado en 1969 por Ken Thompson y Dennis M. Ritchie en los Laboratorios Bell basándose en los lenguajes BCPL y B. Al igual que sus dos predecesores, es un lenguaje orientado a la implementación de Sistemas Operativos (los sistemas operativos Linux y UNIX están escritos mayormente en C), pero se ha convertido en un lenguaje de propósito general de los más usados.

Se trata de un lenguaje no fuertemente tipado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Un ejemplo es la posibilidad de mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos. Destaca su gran riqueza de operadores y expresiones.

Existe un estándar ISO de 1986 denominado ANSI C. En teoría, un lenguaje 100% ANSI C sería portable entre plataformas y/o arquitecturas pero en la práctica esto no es siempre cierto.

# Ventajas e inconvenientes

- **Ventajas:**
  - Es un lenguaje muy eficiente puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas.
  - A pesar de su bajo nivel es el lenguaje más portado en existencia, habiendo compiladores para casi todos los sistemas conocidos.
  - Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado no llevado al extremo (permitiendo ciertas licencias rupturistas).
  - Proporciona facilidades para realizar programas modulares y/o utilizar código o bibliotecas existentes.
- **Inconvenientes:**
  - La portabilidad de los programas escritos en C suele estar limitada seriamente por las diferencias entre sistemas operativos y compiladores.
  - Su flexibilidad y/o la optimización excesiva puede generar programas poco legibles y difíciles de mantener.
  - Algunas de sus posibilidades mal empleadas pueden inducir errores o bugs difíciles de detectar y corregir.
  - Carece de facilidades que se consideran básicas en otros lenguajes, como manejo nativo de cadenas de caracteres. Su núcleo es muy reducido (únicamente expresiones, sentencias condicionales y bucles) mientras que la mayor parte de la funcionalidad se proporciona con bibliotecas externas (la mayor parte bibliotecas estándar y las bibliotecas de la API del sistema). Otra limitación importante es la ausencia de bibliotecas estándar multiplataforma para interfaces gráficas de usuario, multitarea o redes, aunque existen multitud de librerías que suplen estas limitaciones.

## Variantes

Desde el inicio del lenguaje han surgido varias ramas de evolución que han generado varios lenguajes:

- Objective-C es un primer intento de proporcionar soporte para la programación orientada a objetos en C, de escasa difusión, pero actualmente usado en Mac OS X y GNUstep.
- C++ diseñado por Bjarne Stroustrup fue el segundo intento de proporcionar orientación a objetos a C y es la variante más difundida y aceptada. Esta versión combina la flexibilidad y el acceso de bajo nivel de C con las características de la programación orientada a objetos como abstracción, encapsulación y ocultación.

- **C#** (pronunciado *C Sharp* en inglés y de distintas formas en castellano (*C almohadilla, sostenido...*), aunque se suele usar la pronunciación inglesa) es un lenguaje derivado de C/C++ y **Java** desarrollado por **Microsoft**.

## Proceso de compilación

La **compilación** de un programa C se realiza en varias fases que normalmente son automatizadas y ocultadas por los entornos de desarrollo:

1. **Preprocesado** consistente en modificar el **código fuente** en C según una serie de instrucciones (denominadas **directivas de preprocesado**) simplificando de esta forma el trabajo del **compilador**. Por ejemplo, una de las acciones más importantes es la modificación de las inclusiones (`#include`) por las declaraciones reales existentes en el fichero indicado.
2. **Compilación** que genera el **código objeto** a partir del código ya preprocesado.
3. **Enlazado** que une los **códigos objeto** de los distintos módulos y bibliotecas externas (como las **bibliotecas del sistema**) para generar el programa ejecutable final.

### Ejemplo de código C

Sigue el clásico ejemplo "Hola Mundo!" en C:

```
#include <stdio.h> /* Entrada / salida estándar */

int main(void)
{
    puts("¡Hola, Mundo!");

    return 0;
}
```

## ALGOL

Su nombre se debe a la abreviación de *Algorithmic Language*, es un lenguaje muy poderoso que surgió en la década de 1960. El diseño estuvo a cargo de un comité internacional con sede en Europa y fue el primero de los lenguajes con sintaxis definida de manera formal y matemática por lo cual se considera como el precursor de la familia de lenguajes de programación estructurada, otra de sus características es que el manejo de la memoria es dinámico.

Un programa en ALGOL consta de varios *procedures*, que son módulos que pueden ser llamados recursivamente o estar anidados. Por el tipo de manejo de la memoria, las dimensiones de los arreglos pueden variar durante la ejecución del programa.

ALGOL tiene varias versiones y la mayoría de sus aplicaciones son principalmente científicas.

## LOGO

LOGO fue diseñado por un educador y matemático del MIT (Instituto de Tecnología de Massachusetts), Seymour Papert. Es un lenguaje de programación desarrollado para la enseñanza de los niños; quizá es un lenguaje muy simple, pero es muy poderoso porque incorpora el concepto de procedimientos ayudando a los niños a pensar algorítmicamente (lógica y ordenadamente).

## PASCAL

PASCAL es un lenguaje diseñado por Niklaus Wirth, del Instituto Tecnológico de Zurich, en Suiza. Es un lenguaje de programación ampliamente difundido cuyo principal objetivo es enseñar la programación estructurada como una disciplina sistemática, incorporando las siguientes estructuras de control:

- Secuencia.
- Decisión.
- Repetición.

Cuenta además con las siguientes estructuras de datos:

- Arreglos.
- Registros.
- Archivos.
- Conjuntos.
- Tipos de datos definidos por el usuario.

# **FORTRAN**

FORTRAN(FORMula TRANslation) es un lenguaje desarrollado en 1957 por John Backus, de IBM, cuyas versiones más conocidas son la de FORTRAN IV y FORTRAN 77 (FORTRAN Estructurado). Recordemos que la computadora nació de la necesidad de resolver problemas matemáticos y científicos.

A mediados de la década de los 60's FORTRAN pasó a ser ampliamente conocido y usado en buen número de máquinas, dando como resultado una variedad de versiones de las cuales las más importantes fueron FORTRAN II y FORTRAN IV, ambos de IBM.

**Las ventajas de este lenguaje de programación es que maneja variables de doble precisión, variables complejas y modularidad o subrutinas. Fortran**

De Wikipedia, la enciclopedia libre.

Fortran (o más bien FORTRAN hasta principios de los años 90) es un lenguaje de programación desarrollado en los años 50 y activamente utilizado desde entonces. Acrónimo de "Formula Translation".

Fortran se utiliza principalmente en aplicaciones científicas y análisis numérico. Desde 1958 ha pasado por varias versiones, entre las que destacan FORTRAN II, FORTRAN IV, FORTRAN 77 y FORTRAN 90. Si bien el lenguaje era inicialmente un lenguaje imperativo, las últimas versiones incluyen elementos de la programación orientada a objetos.

## ***Tabla de contenidos***

[esconder]

- 1 Evolución del lenguaje
- 2 Principales características
- 3 Especificaciones
- 4 Ejemplo de código
- 5 Enlaces externos
  - 5.1 Recursos de programación

## ***Evolución del lenguaje***

El primer compilador de FORTRAN se desarrolló para una IBM 704 entre 1954 y 1957 por la empresa IBM, por un grupo liderado por John W. Backus. En la época se consideró imprescindible que los programas escritos en FORTRAN corrieran a velocidad comparable a la del lenguaje ensamblador; de otra forma, *nadie lo tomaría en cuenta*.

El lenguaje ha sido ampliamente adoptado por la comunidad científica para escribir aplicaciones con cómputos intensivos. La inclusión en el lenguaje de la aritmética de números complejos amplió la gama de aplicaciones para las cuales el lenguaje se adapta especialmente y muchas técnicas de compilación de lenguajes han sido creadas para mejorar la calidad del código generado por los compiladores de Fortran.

## ***Principales características***

El lenguaje fue diseñado tomando en cuenta que los programas serían escritos en tarjetas perforadas de 80 columnas. Así por ejemplo, las líneas debían ser numeradas y la única alteración posible en el orden de ejecución era producida con la instrucción goto. Estas características han evolucionado de versión en versión. Las versiones actuales contienen subprogramas, recursión y una variada gama de estructuras de control.

[editar]

### **Especificaciones**

Existen dos versiones normalizadas del lenguaje.

- ANSI X3.198-1992 (R1997). Título: *Programming Language "Fortran" Extended*. Conocida como Fortran 90. Se trata de un standard publicado por ANSI.
- ISO/IEC 1539-1:1997. Title: *Information technology - Programming languages - Fortran - Part 1: Base language*. Conocida como Fortran 95. también adoptada por ANSI.

[editar]

### **Ejemplo de código**

```
PROGRAM HOLAMUNDO
  PRINT *, '¡Hola, mundo!'
END
```

# C++

C++ (pronunciado /ce.mas.más/, /si.plos.plós/ o /si.plas.plás/) es un lenguaje de programación, diseñado a mediados de los ochenta, por Bjarne Stroustrup, como extensión del lenguaje de programación C.

Es un lenguaje híbrido, que se puede compilar y resulta más sencillo de aprender para los programadores que ya conocen C. Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes como ROOT (enlace externo). Las principales características del C++ son abstracción (encapsulación), el soporte para programación orientada a objetos (polimorfismo) y el soporte de plantillas o programación genérica (*Templates*). Por ende, se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: La programación estructurada, la programación genérica y la programación orientada a objetos. Las plantillas se las define de la manera siguiente: `template <parámetros> declaración x y se las instancia con x<parámetros>`.

Pero añade otra serie de propiedades que se encuentran más difícilmente en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores
- Identificación de tipos en tiempo de ejecución (*RTTI*)

C++ está considerado por muchos como el lenguaje más potente debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que dificulta mucho su aprendizaje.

El nombre C++ fue propuesto por Rick Masciatti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, "C++" significa "uno más de C" y se refiere a que C++ es una extensión de C.

Algunos dicen que "C++" todavía significa "C", porque "++" en este caso es el operador de la postincrementación, es decir, aumenta el valor de la expresión a la que se refiere, después, en las instrucciones siguientes. Por esto el valor de la expresión en este momento permanece original.

# PHP



El fácil uso y la similaridad con los más comunes lenguajes de programación estructurada, como el C y el Perl, permiten a la mayoría de los programadores experimentados crear aplicaciones complejas con una curva aprendizaje muy suave. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones y prácticas.

Debido al diseño de PHP, también es posible crear aplicaciones con una interfaz gráfica para el usuario o GUI, utilizando la PHP-GTK.

También puede ser usado desde la Línea de comandos, como Perl o Python.

Su interpretación y ejecución se da en el servidor en el cual se encuentra almacenada la página y el cliente solo recibe el resultado de la ejecución. Cuando el cliente hace una petición al servidor para que le envíe una página web, enriquecida con código PHP, el servidor interpretará las instrucciones mezcladas en el cuerpo de la página y las sustituirá con el resultado de la ejecución antes de enviar el resultado a la computadora del cliente. Además es posible utilizarlo para generar archivos PDF, Flash o JPG, entre otros.

Permite la conexión a numerosas bases de datos de forma nativa tales como MySQL, Postgres, Oracle, ODBC, IBM DB2, Microsoft SQL Server y SQLite, lo cual permite la creación de Aplicaciones web muy robustas.

PHP tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos tales como UNIX, Linux, Windows y Mac OS X, y puede interactuar con los servidores de web más populares.

El modelo PHP puede ser visto como una alternativa al sistema de Microsoft que utiliza ASP.NET/C#/VB.NET, a ColdFusion de la compañía Macromedia, a JPS/Java de Sun Microsystems, y al famoso CGI/Perl. Aunque su creación y desarrollo se da en el ámbito de los sistemas libres, bajo la licencia GNU, existe además un compilador comercial denominado Zend Optimizer.

## Historia

PHP fue originalmente diseñado en Perl, seguidos por la escritura de un grupo de CGI binarios escritos en el lenguaje C por el programador Danés-Canadiense Rasmus Lerdorf en el año 1994 para mostrar su currículum vitae y guardar ciertos datos, como la cantidad de tráfico que su página web recibía. El 8 de junio del 1995 fue publicado "Personal Home Page Tools" luego de que Lerdorf lo combinara con su propio *Form Interpreter* para crear PHP/FI.

Dos programadores israelíes de Technion, Zeev Suraski y Andi Gutmas, reescribieron el analizador gramatical (*parser* en inglés) en el año 1997 y crearon la base del PHP 3, cambiando el nombre del lenguaje a la forma actual. Experimentaciones públicas de PHP 3 comenzaron inmediatamente y fue lanzado oficialmente en junio del 1998.

Para el 1999, Suraski y Gutmans reescribieron el código de PHP, produciendo lo que hoy se conoce como Zend Engine o motor Zend. También conformaron a Zend Technologies en Ramat Gan, Israel. En mayo del 2000 PHP 4 fue lanzado bajo el poder del motor Zend Engine 1.0. El 13 de julio del 2004, PHP 5 fue lanzado, utilizando el motor Zend Engine II (o Zend Engine 2). La versión más reciente de PHP es la 5.1, que aún se encuentra en estado beta, que incluye el novedoso PDO (Objetos de Información de PHP o PHP Data Objects) y mejoras utilizando las ventajas que provee el nuevo Zend Engine 2.

## Usos de PHP

Los principales usos del PHP son los siguientes:

- Programación de páginas web dinámicas, habitualmente en combinación con el motor de base datos MySQL, aunque cuenta con soporte nativo para otros motores, incluyendo el estándar ODBC, lo que amplía en gran medida sus posibilidades de conexión.
- Programación en consola, al estilo de Perl, en Linux, Windows y Macintosh.
- Creación de aplicaciones gráficas independientes del navegador, por medio de la combinación de PHP y GTK (GIMP Tool Kit), que permite desarrollar aplicaciones de escritorio tanto para los sistemas operativos basados en Unix, como para Windows y Mac OS X.

# Ventajas de PHP

- Capacidad de acceder la mayoría de las base de datos que se utilizan en la actualidad.
- Leer los datos desde diferentes fuentes, incluyendo datos que pueden meter los usuarios desde formas HTML y manipularlos de forma sencilla.
- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados ext's o extensiones).
- Posee una muy buena documentación en su página oficial ([\[1\]](#)).
- Es Libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.

## Ejemplo de Código PHP

A continuación un ejemplo de una página web sencilla desarrollada utilizando el lenguaje PHP:

```
<html>
<head>
  <title>Ejemplo</title>
</head>
<body>
<?php
if (isset($_POST['muestra'])) {
    echo 'Hola, ' . htmlentities($_POST['nombre'])
        . ', tu comida favorita es: ' . htmlentities($_POST['comida']);
} else {
?>
<form method="POST" action="?">
  ¿Cuál es tu nombre?
  <input type="text" name="nombre"/>
  ¿Cuál es tu comida favorita?
  <select name="comida">
    <option>Spaghetti</option>
    <option>Asado</option>
    <option>Pizza</option>
  </select>
  <input type="submit" name="muestra" value="Seguir">
</form>
<?php
}
?>
</body>
</html>
```

En este código es posible observar las siguientes características:

- Las variables enviadas por un formulario utilizando el método POST, son recibidas en el lenguaje dentro del arreglo `$_POST`, lo cual facilita la obtención de este tipo de datos. Este mismo método es utilizado por el lenguaje para todas las fuentes de información en una aplicación web, tales como cookies en el arreglo `$_COOKIES`, variables de URL en `$_GET`,

variables de sesión utilizando `$_SESSION`, y variables del servidor y del cliente por medio del arreglo `$_SERVER`.

- El código PHP está "encajado" dentro del HTML e interactúa con el mismo, lo que permite diseñar la página Web en un editor común de HTML y añadir el código dinámico dentro de las etiquetas `<?php ?>`.
- El resultado muestra y oculta ciertas porciones del código HTML en forma condicional.
- Es posible utilizar funciones propias del lenguaje para aplicaciones Web como `htmlspecialchars()`, que convierte los caracteres que tienen algún significado especial en el mercado HTML o que podrían desplegarse erróneamente en el navegador como acentos o diéresis, en sus equivalentes en formato HTML.

## Aplicaciones desarrolladas con PHP

- BerriBlog
- Burning Board
- CMSformE
- Drupal
- eZ publish
- Gallery Project
- Geeklog
- Mambo Open Source MOS
- MediaWiki (desarrollado para Wikipedia)
- Moodle
- Phorum
- phpBB
- phpMyAdmin
- PHP-Nuke
- phpPgAdmin
- PhpWiki
- PmWiki
- PostNuke
- Smarty
- SPIP
- vBulletin
- WordPress
- Xaraya
- Xoops

## C Sharp

De Wikipedia, la enciclopedia libre.

C# (pronunciado "ci sharp", "ce agudo", "ce sharp", o -incorrectamente- "ce gato" o "ce almohadilla") es un lenguaje de programación orientado a objetos desarrollado por Microsoft y estandarizado, como parte de su plataforma .NET.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes. C# fue diseñado para combinar el control a bajo nivel de lenguajes como C y la velocidad de programación de lenguajes como Visual Basic.

La pronunciación del nombre viene de la terminología musical, donde C# significa, "do sostenido" (C corresponde a do en la terminología musical anglo-sajona). El símbolo # viene de sobreponer "++" sobre "++" y eliminar las separaciones, indicando así su descendencia de C++.

C#, como parte de la plataforma .NET, está normalizado por ECMA desde diciembre de 2001 (ECMA-334 "Especificación del Lenguaje C#"). En la actualidad se encuentra en desarrollo la versión 2.0 del lenguaje que incluye mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y tipos anulables.

Aunque C# forma parte de la plataforma .NET, ésta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Aunque aún no existen, es posible implementar compiladores que no generen programas para dicha plataforma, sino para una plataforma diferente como Win32 o UNIX.

En la actualidad existen los siguientes compiladores para el lenguaje C#:

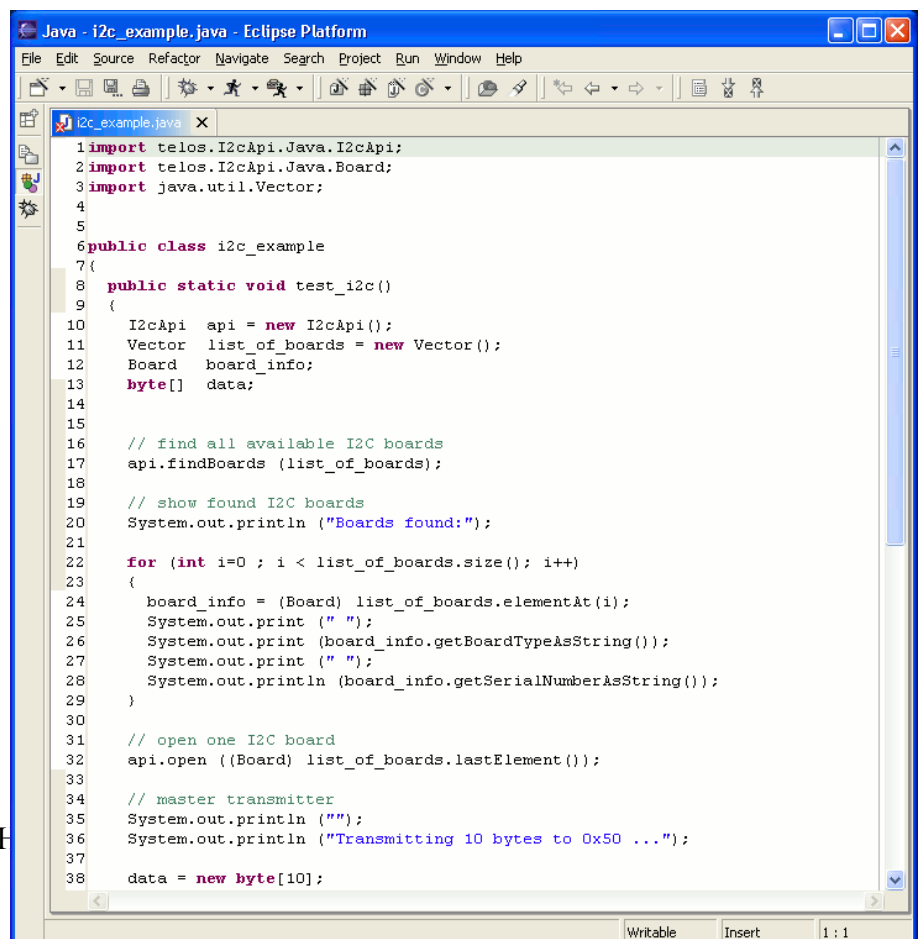
- Microsoft Visual Studio versión 2002, 2003 y próximamente 2005.
- Mono, es una implementación GPL de todo el entorno .NET desarrollado por Ximian. Como parte de esta implementación se incluye un compilador de C#.
- Delphi 2005, de Borland Software Corporation.
- dotGNU Portable.NET, de la Fundación de Programas Libres.

## Lenguaje de programación Java

De Wikipedia, la enciclopedia libre.

Java es una plataforma de software desarrollada por Sun Microsystems, de tal

POR GUSTAVO & NACH



```
1 import telos.I2cApi.Java.I2cApi;
2 import telos.I2cApi.Java.Board;
3 import java.util.Vector;
4
5
6 public class i2c_example
7 {
8     public static void test_i2c()
9     {
10        I2cApi api = new I2cApi();
11        Vector list_of_boards = new Vector();
12        Board board_info;
13        byte[] data;
14
15
16        // find all available I2C boards
17        api.findBoards (list_of_boards);
18
19        // show found I2C boards
20        System.out.println ("Boards found:");
21
22        for (int i=0 ; i < list_of_boards.size(); i++)
23        {
24            board_info = (Board) list_of_boards.elementAt(i);
25            System.out.print (" ");
26            System.out.print (board_info.getBoardTypeAsString());
27            System.out.print (" ");
28            System.out.println (board_info.getSerialNumberAsString());
29        }
30
31        // open one I2C board
32        api.open ((Board) list_of_boards.lastElement());
33
34        // master transmitter
35        System.out.println ("");
36        System.out.println ("Transmitting 10 bytes to 0x50 ...");
37
38        data = new byte[10];
```

manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales.

La *plataforma Java* consta de las siguientes partes:

- El lenguaje de programación, mismo.
- La máquina virtual de Java o JRE, que permite la portabilidad en ejecución.
- El API Java, una biblioteca estándar para el lenguaje.

Originalmente llamado OAK por los ingenieros de Sun Microsystems, Java fue diseñado para correr en computadoras incrustadas. Sin embargo, en 1995, dada la atención que estaba produciendo la Web, Sun Microsystems la distribuyó para sistemas operativos tales como Microsoft Windows.

El lenguaje mismo se inspira en la sintaxis de C++, pero su funcionamiento es más similar al de Smalltalk que a éste. Incorpora sincronización y manejo de tareas en el lenguaje mismo (similar a Ada) e incorpora interfaces como un mecanismo alternativo a la herencia múltiple de C++.

A fines del siglo XX, Java llegó a ser el lenguaje de mayor acogida para programas de servidor. Utilizando una tecnología llamada JSP (similar a otras tecnologías del lado del servidor como ASP de Microsoft o PHP), se hizo muy fácil escribir páginas dinámicas para sitios de Internet. Sumado a esto, la tecnología de JavaBeans, al incorporarse con JSP, permitía adaptar al mundo web el patrón MVC (modelo-vista-controlador) que ya se había aplicado con éxito a interfaces gráficas.

Java llegó a ser extremadamente popular cuando Sun Microsystems introdujo la especificación J2EE (Java 2 Enterprise Edition). Este modelo permite, entre otros, una separación entre la presentación de los datos al usuario (JSP o Applets), el modelo de datos (EJB), y el control (Servlets). Enterprise Java Beans (EJB) es una tecnología de objetos distribuidos que pudo lograr el sueño de muchas empresas como Microsoft e IBM de crear una plataforma de objetos distribuidos con un monitor de transacciones. Con este nuevo estándar, empresas como BEA, IBM, Sun Microsystems, Oracle y otros crearon nuevos "servidores de aplicaciones" que tuvieron gran acogida en el mercado.

Además de programas del servidor, Java permite escribir programas de interfaz gráfica o textual. También se pueden correr programas de manera incorporada o incrustada en los navegadores web de Internet en forma de Java applets, aunque no llegó a popularizarse como se esperaba en un principio.

Los programas en Java generalmente son compilados a un lenguaje intermedio llamado bytecode, que luego son interpretados por una máquina virtual (JVM). Esta última sirve como una plataforma de abstracción entre la máquina y el lenguaje permitiendo que se pueda "escribir el programa una vez, y correrlo en cualquier lado". También existen compiladores nativos de Java, tanto software libre como no libre. El compilador GCC de GNU compila Java a código de máquina con algunas limitaciones al año 2002.

Con la evolución de las diferentes versiones, no sólo se han producido cambios en el lenguaje, sino que se han producido cambios mucho más importantes en sus bibliotecas asociadas, que han pasado de unos pocos cientos en Java 1.0, a más de tres mil en Java 5.0. En particular, se han añadido APIs completamente nuevas, tales como Swing y Java2D.

## Ejemplo de programa en Java

```
import javax.swing.JFrame;
import javax.swing.JLabel;

public class HolaMundo extends JFrame {

    public static void main(String[] args) {
        System.out.println("Vamos a crear una ventana que
salude al mundo.");
        new HolaMundo();
    }

    public HolaMundo() {
        super("Prueba de Java");
        // creo la etiqueta
        JLabel etiqueta = new JLabel("¡Hola, Mundo!");

        etiqueta.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        getContentPane().add(etiqueta);
        setSize(400,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
        setResizable(false);
    }
}
```

## Lenguajes de Programación

Al desarrollarse las primeras computadoras electrónicas, se vio la necesidad de programarlas, es decir, de almacenar en memoria la información sobre la tarea que iban a ejecutar. Las primeras se usaban como calculadoras simples; se les indicaban los pasos de cálculo, uno por uno.

John Von Neumann desarrolló el modelo que lleva su nombre, para describir este concepto de "programa almacenado". En este modelo, se tiene una abstracción de la memoria como un conjunto de celdas, que almacenan simplemente números. Estos números pueden representar dos cosas: los datos, sobre los que va a trabajar el programa; o bien, el programa en sí.

¿Cómo es que describimos un programa como números? Se tenía el problema de representar las acciones que iba a realizar la computadora, y

que la memoria, al estar compuesta por switches correspondientes al concepto de bit, solamente nos permitía almacenar números binarios.

La solución que se tomó fue la siguiente: a cada acción que sea capaz de realizar nuestra computadora, asociarle un número, que será su código de operación (opcode) . Por ejemplo, una calculadora programable simple podría asignar los opcodes :

1 = SUMA, 2 = RESTA, 3 = MULTIPLICA, 4 = DIVIDE.

Supongamos que queremos realizar la operación  $5 * 3 + 2$ , en la calculadora descrita arriba. En memoria, podríamos "escribir" el programa de la siguiente forma:

Localidad	Opcod	Significado	Comentario
0	5	5	En esta localidad, tenemos el primer número de la fórmula
1	3	*	En esta localidad, tenemos el opcode que representa la multiplicación.
2	3	3	En esta localidad, tenemos el segundo número de la fórmula
3	1	+	En esta localidad, tenemos el opcode que representa la suma.
4	2	2	En esta localidad, tenemos el último número de la fórmula

Podemos ver que con esta representación, es simple expresar las operaciones de las que es capaz el hardware (en este caso, nuestra calculadora imaginaria), en la memoria.

La descripción y uso de los opcodes es lo que llamamos lenguaje de máquina . Es decir, la lista de códigos que la máquina va a interpretar como instrucciones, describe las capacidades de programación que tenemos de ella; es el lenguaje más primitivo, depende directamente del hardware, y requiere del programador que conozca el funcionamiento de la máquina al más bajo nivel.

los lenguajes más primitivos fueron los lenguajes de máquina. Esto, ya que el hardware se desarrolló antes del software, y además cualquier software finalmente tiene que expresarse en el lenguaje que maneja el hardware.

La programación en esos momentos era sumamente tediosa, pues el programador tenía que "bajarse" al nivel de la máquina y decirle, paso a pasito, cada punto de la tarea que tenía que realizar. Además, debía expresarlo en forma numérica; y por supuesto, este proceso era propenso a errores, con lo que la productividad del programador era muy limitada. Sin embargo, hay que recordar que en estos momentos, simplemente aún no existía alternativa.

El primer gran avance que se dio, como ya se comentó, fue la abstracción dada por el Lenguaje Ensamblador, y con él, el nacimiento de las primeras herramientas automáticas para generar el código máquina. Esto redujo los errores triviales, como podía ser el número que correspondía a una operación, que son sumamente engorrosos y difíciles de detectar,

pero fáciles de cometer. Sin embargo, aún aquí es fácil para el programador perderse y cometer errores de lógica, pues debe bajar al nivel de la forma en que trabaja el CPU, y entender bien todo lo que sucede dentro de él.

Con el desarrollo en los 50s y 60s de algoritmos de más elevado nivel, y el aumento de poder del hardware, empezaron a entrar al uso de computadoras científicos de otras ramas; ellos conocían mucho de Física, Química y otras ramas similares, pero no de Computación, y por supuesto, les era sumamente complicado trabajar con lenguaje Ensamblador en vez de fórmulas. Así, nació el concepto de Lenguaje de Alto Nivel, con el primer compilador de FORTRAN (FORmula TRANslation), que, como su nombre indica, inició como un "simple" esfuerzo de traducir un lenguaje de fórmulas, al lenguaje ensamblador y por consiguiente al lenguaje de máquina. A partir de FORTRAN, se han desarrollado innumerables lenguajes, que siguen el mismo concepto: buscar la mayor abstracción posible, y facilitar la vida al programador, aumentando la productividad, encargándose los compiladores o intérpretes de traducir el lenguaje de alto nivel, al lenguaje de computadora.

Hay que notar la existencia de lenguajes que combinan características de los de alto nivel y los de bajo nivel (es decir, Ensamblador). Mi ejemplo favorito es C: contiene estructuras de programación de alto nivel, y la facilidad de usar librerías que también son características de alto nivel; sin embargo, fue diseñado con muy pocas instrucciones, las cuales son sumamente sencillas, fáciles de traducir al lenguaje de la máquina; y requiere de un entendimiento apropiado de cómo funciona la máquina, el uso de la memoria, etcétera. Por ello, muchas personas consideramos a lenguajes como C (que fue diseñado para hacer sistemas operativos), lenguajes de nivel medio.

## ***QBasic***

Qbasic es un lenguaje de alto nivel, el cual consiste en instrucciones que los humanos pueden relacionar y entender. El compilador de Qbasic se encarga de traducir el mismo a lenguaje de máquina.

Un programa es una secuencia de instrucciones. El proceso de ejecutar esas instrucciones se llama correr el programa. Los programas contienen las funciones de entrada, procesamiento y salida. La persona que resuelve problemas mediante escribir programas en la computadora se conoce como programador. Después de analizar el problema y desarrollar un plan para solucionarlo, escribe y prueba el programa que instruye a la computadora como llevar a cabo el plan. El procedimiento que realiza el programador se define como "problem solving". Pero es necesario especificar que un programador y un usuario no son lo mismo. Un usuario es cualquier persona que use el programa.

**Ejemplo de qbasic, para hacer una calculadora**

**DIM total AS DOUBLE**

**DIM number AS DOUBLE**

**DIM secondNumber AS DOUBLE**

**DIM more AS STRING**

**DIM moreNumbers AS STRING**

**DIM operation AS STRING**

**total = 0**

**more = "y"**

**moreNumbers = "c"**

**CLS**

**WHILE more = "y"**

**INPUT "Enter the first number"; number**

**total = number**

**WHILE moreNumbers = "c"**

**COLOR 14**

**PRINT "The total is:"; total**

**COLOR 7**

**PRINT "Select an operation"**

**COLOR 2**

**PRINT "(+)"**

**COLOR 5**

**PRINT "(-)"**

**COLOR 1**

**PRINT "(x)"**

```
COLOR 4  
INPUT "("; operation  
COLOR 7  
CLS  
IF operation = "+" THEN  
REM where we do additions  
PRINT "Enter the number to Add to"; total  
INPUT secondNumber  
total = secondNumber + total  
COLOR 14  
PRINT "The total is now:"; total  
COLOR 7  
ELSE  
IF operation = "-" THEN  
REM subtraction  
PRINT "Enter the number to Subtract from"; total  
INPUT secondNumber  
total = total - secondNumber  
COLOR 14  
PRINT "The total is now:"; total  
COLOR 7  
ELSE  
IF operation = "x" THEN  
REM multiplication  
PRINT "Enter the number to Multiply"; total; "by"
```

```
INPUT secondNumber
total = secondNumber * total
REM * is the multiplication sign in programs
COLOR 14
PRINT "The total is now:"; total
COLOR 7
ELSE
IF operation = "/" THEN
REM division
PRINT "Enter the number to Divide"; total; "by"
INPUT secondNumber
IF secondNumber = 0 THEN
COLOR 4
PRINT "You cannot divide by zero"
COLOR 7
ELSE
total = total / secondNumber
REM / is the division sign in programs
END IF
COLOR 14
PRINT "The total is now:"; total
COLOR 7
ELSE
PRINT "you must select an operation"
END IF
```

```

END IF
END IF
END IF
INPUT "Do you wish to continue (c) or start with new numbers
(n)";moreNumbers
CLS
WEND
COLOR 14
PRINT "The grand total is:"; total
COLOR 7
INPUT "Do you wish to make more calculations (y - n)"; more
moreNumbers = "c"
REM if we don't put "moreNumbers" back to y, it will always
REM come back to "Do you wish to make more calculations" and never
REM ask
for numbers again
REM (try it)
total = 0
REM if we don't reset the total to 0, it will just
REM keep on adding to the total
WEND
END

```

## ***Linux***

Linux es una implementación del sistema operativo UNIX (uno más de entre los numerosos clónicos del histórico Unix), pero con la originalidad de ser gratuito y a la vez muy potente, que sale muy bien parado (no pocas veces victorioso) al compararlo con las versiones comerciales para

sistemas de mayor envergadura y por tanto teóricamente superiores. Comenzó como proyecto personal del –entonces estudiante- Linus Torvalds, quien tomó como punto de partida otro viejo conocido, el Minix de Andy. S. Tanenbaum (profesor de sistemas operativos que creó su propio sistema operativo Unix en PCs XT para usarlo en su docencia). Actualmente Linus lo sigue desarrollando, pero a estas alturas el principal autor es la red Internet, desde donde una gigantesca familia de programadores y usuarios aportan diariamente su tiempo aumentando sus prestaciones y dando información y soporte técnico mútuo. La versión original -y aun predominante- comenzó para PCs compatibles (Intel 386 y superiores), existiendo también en desarrollo versiones para prácticamente todo tipo de plataformas:

De todas ellas la más reciente en este momento es la versión para PowerMac <<http://www.mklinux.org>> (el PowerPC de Apple) basada en el microkernel Mach 3.0 y de la que ya hay una distribución para desarrolladores avalada directamente por Apple y OSF pero conservando el espíritu (gratuito, de libre distribución, etc) de la versión original. Un servidor la acaba de probar hace unos días y se ha llevado una grata sorpresa (aún tendrá muuuchos fallos, pero para ser una primerísima versión y el poco tiempo que lleva en marcha, ha avanzado más de lo que me esperaba).

## Ensamblador

Cuando abstraemos los opcodes y los sustituimos por una palabra que sea una clave de su significado, a la cual comúnmente se le conoce como mnemónico , tenemos el concepto de Lenguaje Ensamblador . Así, podemos definir simplemente al Lenguaje Ensamblador de la siguiente forma:

Lenguaje Ensamblador es la primera abstracción del Lenguaje de Máquina , consistente en asociar a los opcodes palabras clave que faciliten su uso por parte del programador

Como se puede ver, el Lenguaje Ensamblador es directamente traducible al Lenguaje de Máquina, y viceversa; simplemente, es una abstracción que facilita su uso para los seres humanos. Por otro lado, la computadora no entiende directamente al Lenguaje Ensamblador; es necesario traducirle a Lenguaje de Máquina. Originalmente, este proceso se hacía a mano, usando para ello hojas donde se escribían tablas de programa similares al ejemplo de la calculadora que vimos arriba . Pero, al ser tan directa la traducción, pronto aparecieron los programas Ensambladores, que son traductores que convierten el código fuente (en Lenguaje Ensamblador) a código objeto (es decir, a Lenguaje de Máquina).

Una característica que hay que resaltar, es que al depender estos lenguajes del hardware, hay un distinto Lenguaje de Máquina (y, por

consiguiente, un distinto Lenguaje Ensamblador) para cada CPU. Por ejemplo, podemos mencionar tres lenguajes completamente diferentes, que sin embargo vienen de la aplicación de los conceptos anteriores:

1.Lenguaje Ensamblador de la familia Intel 80x86 2.Lenguaje Ensamblador de la familia Motorola 68000 3.Lenguaje Ensamblador del procesador POWER, usado en las IBM RS/6000.

Tenemos 3 fabricantes distintos, compitiendo entre sí y cada uno aplicando conceptos distintos en la manufactura de sus procesadores, su arquitectura y programación; todos estos aspectos, influyen en que el lenguaje de máquina y ensamblador cambie bastante.

### Ventajas y desventajas del Lenguaje Ensamblador

Una vez que hemos visto la evolución de los lenguajes, cabe preguntarse: ¿En estos tiempos "modernos", para qué quiero el Lenguaje Ensamblador?

El proceso de evolución trajo consigo algunas desventajas, que ahora veremos como las ventajas de usar el Lenguaje Ensamblador, respecto a un lenguaje de alto nivel:

1.Velocidad

2.Eficiencia de tamaño

3.Flexibilidad

Por otro lado, al ser un lenguaje más primitivo, el Ensamblador tiene ciertas desventajas respecto a los lenguajes de alto nivel:

1.Tiempo de programación 2.Programas fuente grandes 3.Peligro de afectar recursos inesperadamente 4.Falta de portabilidad

### Velocidad

El proceso de traducción que realizan los intérpretes, implica un proceso de cómputo adicional al que el programador quiere realizar. Por ello, nos encontraremos con que un intérprete es siempre más lento que realizar la misma acción en Lenguaje Ensamblador, simplemente porque tiene el costo adicional de estar traduciendo el programa, cada vez que lo ejecutamos.

De ahí nacieron los compiladores, que son mucho más rápidos que los intérpretes, pues hacen la traducción una vez y dejan el código objeto, que ya es Lenguaje de Máquina, y se puede ejecutar muy rápidamente. Aunque el proceso de traducción es más complejo y costoso que el de ensamblar un programa, normalmente podemos despreciarlo, contra las ventajas de codificar el programa más rápidamente.

Sin embargo, la mayor parte de las veces, el código generado por un compilador es menos eficiente que el código equivalente que un programador escribiría. La razón es que el compilador no tiene tanta inteligencia, y requiere ser capaz de crear código genérico, que sirva tanto para un programa como para otro; en cambio, un programador humano puede aprovechar las características específicas del problema, reduciendo la generalidad pero al mismo tiempo, no desperdicia ninguna instrucción, no hace ningún proceso que no sea necesario.

Para darnos una idea, en una PC, y suponiendo que todos son buenos programadores, un programa para ordenar una lista tardará cerca de 20 veces más en Visual Basic (un intérprete), y 2 veces más en C (un compilador), que el equivalente en Ensamblador.

Por ello, cuando es crítica la velocidad del programa, Ensamblador se vuelve un candidato lógico como lenguaje.

Ahora bien, esto no es un absoluto; un programa bien hecho en C puede ser muchas veces más rápido que un programa mal hecho en Ensamblador; sigue siendo sumamente importante la elección apropiada de algoritmos y estructuras de datos. Por ello, se recomienda buscar optimizar primero estos aspectos, en el lenguaje que se desee, y solamente usar Ensamblador cuando se requiere más optimización y no se puede lograr por estos medios.

## Tamaño

Por las mismas razones que vimos en el aspecto de velocidad, los compiladores e intérpretes generan más código máquina del necesario; por ello, el programa ejecutable crece. Así, cuando es importante reducir el tamaño del ejecutable, mejorando el uso de la memoria y teniendo también beneficios en velocidad, puede convenir usar el lenguaje Ensamblador. Entre los programas que es crítico el uso mínimo de memoria, tenemos a los virus y manejadores de dispositivos (drivers). Muchos de ellos, por supuesto, están escritos en lenguaje Ensamblador.

## Flexibilidad

Las razones anteriores son cuestión de grado: podemos hacer las cosas en otro lenguaje, pero queremos hacerlas más eficientemente. Pero todos los lenguajes de alto nivel tienen limitantes en el control; al hacer abstracciones, limitan su propia capacidad. Es decir, existen tareas que la máquina puede hacer, pero que un lenguaje de alto nivel no permite. Por ejemplo, en Visual Basic no es posible cambiar la resolución del monitor a medio programa; es una limitante, impuesta por la abstracción del GUI Windows. En cambio, en ensamblador es sumamente sencillo, pues tenemos el acceso directo al hardware del monitor.

Resumiendo, la flexibilidad consiste en reconocer el hecho de que

Todo lo que puede hacerse con una máquina, puede hacerse en el lenguaje ensamblador de esta máquina; los lenguajes de alto nivel tienen en una u otra forma limitantes para explotar al máximo los recursos de la máquina.

### Tiempo de programación

Al ser de bajo nivel, el Lenguaje Ensamblador requiere más instrucciones para realizar el mismo proceso, en comparación con un lenguaje de alto nivel. Por otro lado, requiere de más cuidado por parte del programador, pues es propenso a que los errores de lógica se reflejen más fuertemente en la ejecución.

Por todo esto, es más lento el desarrollo de programas comparables en Lenguaje Ensamblador que en un lenguaje de alto nivel, pues el programador goza de una menor abstracción.

### Programas fuente grandes

Por las mismas razones que aumenta el tiempo, crecen los programas fuentes; simplemente, requerimos más instrucciones primitivas para describir procesos equivalentes. Esto es una desventaja porque dificulta el mantenimiento de los programas, y nuevamente reduce la productividad de los programadores.

### Peligro de afectar recursos inesperadamente

Tenemos la ventaja de que todo lo que se puede hacer en la máquina, se puede hacer con el Lenguaje Ensamblador (flexibilidad). El problema es que todo error que podamos cometer, o todo riesgo que podamos tener, podemos tenerlo también en este Lenguaje. Dicho de otra forma, tener mucho poder es útil pero también es peligroso.

En la vida práctica, afortunadamente no ocurre mucho; sin embargo, al programar en este lenguaje verán que es mucho más común que la máquina se "cuelgue", "bloquee" o "se le vaya el avión"; y que se reinicialice. ¿Por qué?, porque con este lenguaje es perfectamente posible (y sencillo) realizar secuencias de instrucciones inválidas, que normalmente no aparecen al usar un lenguaje de alto nivel.

En ciertos casos extremos, puede llegarse a sobrescribir información del CMOS de la máquina (no he visto efectos más riesgosos); pero, si no la conservamos, esto puede causar que dejemos de "ver" el disco duro, junto con toda su información.

### Falta de portabilidad

Como ya se mencionó, existe un lenguaje ensamblador para cada máquina; por ello, evidentemente no es una selección apropiada de lenguaje cuando deseamos codificar en una máquina y luego llevar los

programas a otros sistemas operativos o modelos de computadoras. Si bien esto es un problema general a todos los lenguajes, es mucho más notorio en ensamblador: yo puedo reutilizar un 90% o más del código que desarrollo en "C", en una PC, al llevarlo a una RS/6000 con UNIX, y lo mismo si después lo llevo a una Macintosh, siempre y cuando esté bien hecho y siga los estándares de "C", y los principios de la programación estructurada. En cambio, si escribimos el programa en Ensamblador de la PC, por bien que lo desarrollemos y muchos estándares que sigamos, tendremos prácticamente que reescribir el 100 % del código al llevarlo a UNIX, y otra vez lo mismo al llevarlo a Mac.