UD01 – ELEMENTOS DE UN PROGRAMA INFORMÁTICO

Contenido

1 Lenguajes de programación. Lenguajes de bajo y alto nivel

1.1 ¿Qué es un programa?

Un primer paso para poder empezar a estudiar cómo hay que hacer un programa informático es tener claro qué es un programa. En contraste con otros términos usados en informática, es posible referirse a un "programa" en el lenguaje coloquial sin tener que estar hablando necesariamente de ordenadores. Os podría estar refiriéndose al programa de un ciclo de conferencias o de cine. Pero, a pesar de no tratarse de un contexto informático, este uso ya os aporta una idea general de su significado: un conjunto de eventos ordenados de forma que suceden de forma secuencial en el tiempo, uno tras otro.

Otro uso habitual, ahora ya sí vinculado al contexto de las máquinas y los autómatas, podría ser para referirse al programa de una lavadora o de un robot de cocina. En este caso, sin embargo, lo que sucede es un conjunto no tanto de eventos, sino de órdenes que el electrodoméstico sigue ordenadamente. Una vez seleccionado el programa que queremos, el electrodoméstico hace todas las tareas correspondientes de manera autónoma.

Entrando ya, ahora sí, en el mundo de los ordenadores, la forma en que se estructura el tipo de tareas que estos pueden hacer tiene mucho en común con los programas de electrodomésticos. En este caso, sin embargo, en lugar de transformar ingredientes (o lavar ropa sucia, si se tratase de una lavadora), lo que la computadora transforma es información o datos. ***Un programa informático no es más que una serie de órdenes que se llevan a cabo secuencialmente, aplicadas sobre un conjunto de datos.***

¿Qué datos procesa un programa informático? Bueno, esto dependerá del tipo de programa:

- Un editor procesa los datos de un documento de texto.
- Una hoja de cálculo procesa datos numéricos.
- Un videojuego procesa los datos que dicen la forma y ubicación de enemigos y jugadores, etc.
- Un navegador web procesa las órdenes del usuario y los datos que recibe desde un servidor en Internet.

Por lo tanto, la tarea de un programador informático es escoger qué órdenes constituirán un programa de ordenador, en qué orden se deben llevar a cabo y sobre qué datos hay que aplicarlas, para que el programa lleve a cabo la tarea que ha de resolver. La dificultad de todo ello será más grande o pequeña dependiendo de la complejidad misma de lo que es necesario que el programa haga. No es lo mismo establecer qué debe hacer el ordenador para resolver una multiplicación de tres números que para procesar textos o visualizar páginas en Internet.

Por "ejecutar un programa" se entiende hacer que el ordenador siga todas sus órdenes, desde la primera hasta la última.

Por otra parte, una vez hecho el programa, cada vez que lo ejecute, el ordenador cumplirá todas las órdenes del programa.

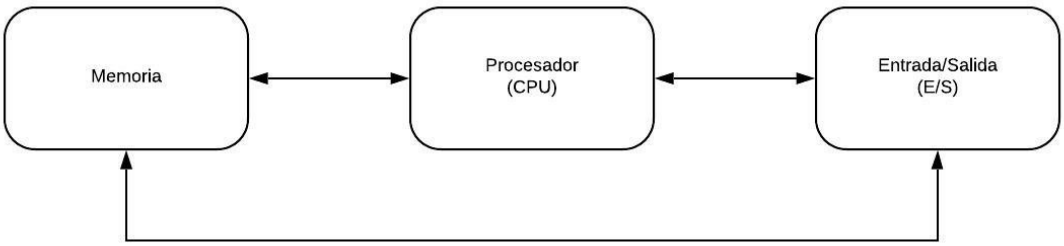
De hecho, un ordenador es incapaz de hacer absolutamente nada por sí mismo, siempre hay que decirle qué debe hacer. Y eso se le llama mediante la ejecución de programas. Aunque desde el punto de vista del usuario puede parecer que cuando se pone en marcha un ordenador este funciona sin ejecutar ningún programa concreto, hay que tener en cuenta que su sistema operativo es un programa que está siempre en ejecución.

1.2 Tipo de órdenes que acepta un ordenador

Para llevar a cabo la tarea encomendada, un ordenador puede aceptar diferentes tipos de órdenes. Estas se encuentran limitadas a las capacidades de los componentes que lo conforman. Por lo tanto, es importante tener presente este hecho para saber qué se puede pedir en el ordenador al crear un programa.

La estructura interna del ordenador se divide en una serie de componentes, todos comunicados entre sí, tal como muestra la figura 1 de manera muy simplista, pero suficiente para empezar. Cada orden de un programa está vinculada de una manera u otra a alguno de estos componentes.

Figura 1 Componentes básicos de un ordenador



El **procesador** (o CPU, *Central Processing Unit*) es el centro neurálgico del ordenador y el elemento que es capaz de llevar a cabo las órdenes de manipulación y transformación de los datos. Un conjunto de datos se puede transformar de muchas maneras, según las capacidades que ofrezca cada procesador. Sin embargo, hay muchas transformaciones que todos pueden hacer. Un ejemplo es la realización de operaciones aritméticas (suma, resta, multiplicación, división), tal como hacen las calculadoras.

La **memoria** permite almacenar datos mientras éstas no están siendo directamente manipuladas por el procesador. Cualquier dato que debe ser tratado por un programa estará en la memoria. Mediante el programa se puede ordenar al procesador que guarde ciertos datos o que los recupere en cualquier momento. Normalmente, cuando se habla de memoria a este nivel nos referimos a memoria dinámica o RAM (*Random Access Memory*, memoria de acceso aleatorio). Esta memoria no es persistente y una vez termina la ejecución del programa todos los datos con las que trataba desvanecen. Por lo tanto, la información no se guardará entre sucesivas ejecuciones diferentes de un mismo programa.

En ciertos contextos es posible que nos encontramos también con memoria ROM (*Read-Only Memory*, memoria de sólo lectura). En ésta, los datos están predefinidas de fábrica y no se puede almacenar nada, sólo podemos leer lo que contiene. Hay que decir que no es el caso más habitual.

El sistema de **entrada / salida** (abreviado como E / S) permite el intercambio de datos con el exterior del ordenador, más allá del procesador y la memoria. Esto permite traducir la información procesada en acciones de control sobre cualquier periférico conectado al ordenador. Un ejemplo típico es establecer una vía de diálogo con el usuario, ya sea por medio del teclado o del ratón para pedirle información, como por la pantalla, para mostrar los resultados del programa. Este sistema es clave para convertir un ordenador en una herramienta de propósito general, ya que lo capacita para controlar todo tipo de aparatos diseñados para conectarse.

Otra posibilidad importante de la computadora, dadas las limitaciones del sistema de memoria, es poder interactuar con el hardware de almacenamiento persistente de datos, como un disco duro.

Partiendo de esta descripción de las tareas que puede llevar a cabo un ordenador según los elementos que lo componen, un ejemplo de programa para multiplicar dos números es el mostrado en la tabla 1. Expresado en lenguaje natural (el lenguaje natural es aquel que empleamos los humanos para comunicarnos habitualmente). Insistir en que los datos deben estar siempre almacenados en la memoria para poder operar.

Tabla1 Un programa que multiplica dos números usando lenguaje natural

Orden	**Elemento que la efectúa**
1. Lee un número del teclado.	E / S (teclado)
2. Guardar el número en la memoria.	memoria
3. Lee otro número del teclado.	E / S (teclado)
4. Guardar el número en la memoria.	memoria
5. Recupera los números de la memoria y haz la multiplicación.	procesador
6. Guardar el resultado en la memoria.	memoria
7. Muestra el resultado en la pantalla.	E / S (pantalla)

1.3 Crear un programa ejecutable

Para crear un programa hay que establecer qué órdenes deben darse en el ordenador y en qué secuencia. Ahora bien, hoy en día los ordenadores todavía no entienden el lenguaje natural (como se utiliza en la tabla 1), ya que está lleno de ambigüedades y aspectos semánticos que pueden depender del contexto.

Para especificar las órdenes que debe seguir un ordenador el que se usa es un **lenguaje de programación**. Se trata de un lenguaje artificial diseñado expresamente para crear algoritmos que puedan ser llevados a cabo por el ordenador. (Por *artificial* entendemos lo que no ha evolucionado a partir del uso entre humanos, sino que ha sido creado expresamente, en este caso para ser usado con los ordenadores).

Existen muchos lenguajes de programación, cada uno con sus características propias, de manera que unos son más o menos indicados para resolver un tipo de tareas u otras.

Los lenguajes de programación están formados por un conjunto de símbolos (llamado alfabeto), reglas gramaticales (léxico/morfológicas y sintácticas) y reglas semánticas, que en conjunto definen las estructuras válidas en el lenguaje y su significado. De esta forma la computadora puede interpretar correctamente cada orden que se le da.

En un lenguaje de programación determinado, el conjunto de órdenes concretas que se pide al ordenador que haga se denomina conjunto de **instrucciones**.

Normalmente, el conjunto de instrucciones de un programa se almacena dentro de un conjunto de archivos. Estos archivos los edita el programador para crear o modificar el programa.

Los lenguajes de programación se pueden clasificar en diferentes categorías según sus características. De hecho, algunas de las propiedades del lenguaje de programación tienen consecuencias importantes sobre el proceso a seguir para poder crear un programa y para ejecutarlo.

Hay dos maneras de clasificar los lenguajes de programación:

- Según si se trata de un lenguaje compilado o interpretado. Esta propiedad afecta los pasos a seguir para llegar a obtener un archivo ejecutable. O sea, un fichero con el mismo formato que el de las aplicaciones que puede tener instaladas en su equipo.
- Según si se trata de un lenguaje de nivel alto o bajo. Esta propiedad indica el grado de abstracción del programa y si sus instrucciones están más o menos estrechamente vinculadas al funcionamiento del hardware de un ordenador.

1.3.1 El lenguaje máquina

El lenguaje de máquina o código de máquina es el lenguaje que elegiríamos si quisiéramos hacer un programa que trabajara directamente sobre el procesador.

En este lenguaje, cada una de las instrucciones se representa con una secuencia binaria, en ceros (0) y unos (1), y todo el conjunto de instrucciones del programa queda almacenado de manera consecutiva dentro de un fichero de datos en binario.

Cuando se pide la ejecución de un programa en lenguaje de máquina, este se carga en la memoria del ordenador. A continuación, el procesador va leyendo una por una cada una de las instrucciones, las decodifica y las convierte en señales eléctricas de control sobre los elementos del ordenador para que hagan la tarea solicitada. A muy bajo nivel, casi se puede llegar a establecer una correspondencia entre los 0 y 1 de cada instrucción y el estado resultante de los transistores* dentro de los chips internos del procesador.

El transistor es el componente básico de un sistema digital. Se puede considerar como un interruptor, donde 1 indica que pasa corriente y 0 que no pasa.

El conjunto de instrucciones que es capaz de decodificar correctamente un procesador y convertir en señales de control es específico para cada modelo y está definido por su fabricante. El diseñador de cada procesador se inventó la sintaxis y las instrucciones del código máquina de acuerdo con sus necesidades cuando diseñó el hardware. Por tanto, las instrucciones en formato binario que puede decodificar un tipo de procesador pueden ser totalmente incompatibles con las que puede decodificar otro. Esto es lógico, ya que sus circuitos son diferentes y, por tanto, las señales eléctricas de control que debe generar son diferentes para cada caso. Dos secuencias de 0 y 1 iguales pueden tener efectos totalmente diferentes en dos procesadores de modelos diferentes,

Un programa escrito en lenguaje de máquina es específico para un tipo de procesador concreto. No se puede ejecutar sobre ningún otro procesador, a menos que sean compatibles. Un procesador concreto sólo entiende directamente el lenguaje de máquina especificado por su fabricante.

Aunque, como se puede ver, en realidad hay muchos lenguajes de máquina diferentes, se usa esta terminología para englobar a todos. Si se quiere concretar más se puede decir "lenguaje de máquina del procesador X".

Ahora bien, estrictamente hablando, si optaran por hacer un programa en lenguaje de máquina, nunca lo haría generando directamente archivos con todo de secuencias binarias. Sólo tiene que imaginar el aspecto de un programa de este tipo en formato impreso, consistente en una enorme ristra de 0 y 1. Sería totalmente incomprensible y prácticamente imposible de analizar. En realidad, lo que se usa es un sistema auxiliar de mnemotécnicos en el que se asigna a cada instrucción en binario un identificador en formato de texto legible, más

fácilmente comprensible para los humanos. De este modo, es posible generar un programa a partir de ficheros en formato texto.

Esta recopilación de mnemotécnicos es lo que se conoce como el **lenguaje ensamblador**.

A título ilustrativo, la tabla 2 muestra las diferencias de aspecto entre un lenguaje de máquina y ensamblador equivalentes para un procesador de modelo 6502. Sin entrar en más detalles, es importante mencionar que tanto en lenguaje de máquina como en ensamblador cada una de las instrucciones se corresponde a una tarea muy simple sobre uno de sus componentes. Hacer que el ordenador realice tareas complejas implica tener que generar muchas instrucciones en estos lenguajes.

Tabla 2 Equivalencia entre un lenguaje ensamblador y su lenguaje de máquina asociado

instrucción ensamblador	**Lenguaje de máquina equivalente**
LDA # 6	1010100100000110
CMP & 3500	1100110100000000000110101
LDA & 70	1010010101110000
INX	11101111

1.3.2 Lenguajes compilados

Para crear un programa lo que haremos es crear un archivo y escribir el conjunto de instrucciones que queremos que el ordenador ejecute. Para empezar, bastará con un editor de texto simple (como el Bloc de Notas (Windows) o Gedit (Linux)).

Una vez se ha terminado de escribir el programa, el conjunto de archivos de texto resultante donde se encuentran las instrucciones se dice que contiene el **código fuente**.

Este sistema de programar más cómodo para los humanos hace surgir un problema, y es que los archivos de código fuente no contienen lenguaje de máquina y, por tanto, resultan incomprensibles para el procesador. No se le puede pedir que la ejecute directamente; esto sólo es posible usando lenguaje de máquina. Para poder generar código máquina hay que hacer un proceso de traducción desde los mnemotécnicos que contiene cada archivo a las secuencias binarias que entiende el procesador.

El proceso llamado ***compilación*** es la traducción del código fuente de los archivos del programa en archivos en formato binario que contienen las instrucciones en un formato que el procesador puede entender. El contenido de estos archivos se denomina **código objeto**. El programa que hace este proceso se denomina **compilador**.

El código objeto de las instrucciones en la tabla 2 tiene este aspecto:

```
1010100100000011011 0011010000000000011 0101101001010111100 0011101111
```

Para el caso del ensamblador el proceso de compilación es muy sencillo, ya que es una mera traducción inmediata de cada mnemotécnico a la secuencia binaria que le corresponde. En principio, con esto ya habría suficiente para poder hacer cualquier programa, pero ceñirse sólo

al uso de lenguaje ensamblador conlleva ciertas ventajas e inconvenientes que hacen que en realidad no sea usado normalmente, sólo en casos muy concretos.

Cabe destacar que con ensamblador el programador tiene control absoluto del hardware del ordenador a nivel muy bajo. Prácticamente se puede decir que controla cada señal eléctrica y los valores de los transistores dentro de los chips. Esto permite llegar a hacer programas muy eficientes en el que el ordenador hace exactamente lo que se le indica sin ningún tipo de ambigüedad. En contraposición, los programas en ensamblador sólo funcionan para un tipo de procesador concreto, no son portables. Si hay que hacer para otra arquitectura, normalmente hay que empezar de cero. Además -y es el motivo de más peso para pensárselo dos veces si se quiere usar este lenguaje- crear un programa complejo requiere un grado enorme de experiencia sobre cómo funciona el hardware del procesador, y la longitud sería considerable. Esto hace que sean programas complicados de entender y que haya que dedicar mucho tiempo a hacerlos.

Se considera que el código de máquina y el ensamblador son los lenguajes de nivel más bajo que existen, ya que sus instrucciones dependen directamente del tipo de procesador.

Actualmente, para generar la inmensa mayoría de programas se utilizan los llamados lenguajes de alto nivel. Estos ofrecen un conjunto de instrucciones que son fáciles de entender para el ser humano y, por tanto, poseen un grado de abstracción más alto que el lenguaje ensamblador (ya que no están vinculados a un modelo de procesador concreto). Cada una de las instrucciones se corresponde a una orden genérica en la que lo más importante es su aspecto funcional (que se quiere hacer), sin importar cómo se materializa esto en el hardware del ordenador ni mucho menos en señales eléctricas. En cualquier caso, hay que advertir que esta clasificación no siempre es absoluta. Se puede decir que hay lenguajes de "nivel más alto o bajo que otros".

El proceso para generar un programa a partir de un lenguaje de nivel alto es muy parecido al que hay que seguir para hacerlo usando el lenguaje ensamblador, ya que las instrucciones también se escriben en formato texto dentro de archivos de código fuente. La ventaja adicional es que el formato y la sintaxis ya no están ligados al procesador, y, por tanto, pueden tener el formato que quiera el inventor del lenguaje sin que deba tener en cuenta el hardware de los ordenadores donde se ejecutará. Normalmente, las instrucciones y la sintaxis han sido elegidas para facilitar la tarea de creación y comprensión del código fuente del programa.

De hecho, en los lenguajes de nivel alto más frecuentes, entre los que está los que se aprenderán a usar en este módulo, las instrucciones dentro de un programa se escriben como una secuencia de sentencias.

Una **sentencia** es el elemento mínimo de un lenguaje de programación, a menudo identificado por una cadena de texto especial, que sirve para describir exactamente una acción que el programa tiene que hacer.

Por tanto, a partir de ahora se usará el término *sentencia* en lugar de *instrucción* cuando el texto se refiera a un lenguaje de este tipo.

2 Ensambladores, compiladores e intérpretes

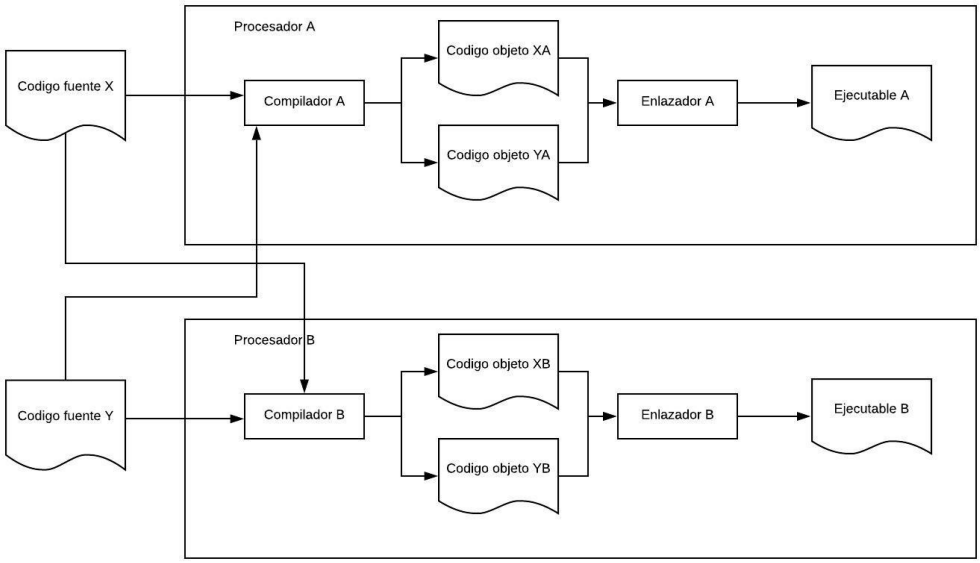
2.1 Lenguajes compilados

Una vez se han terminado de generar los archivos con su código fuente, estos se han de compilar para traducirlos a código objeto.

El proceso de traducción a código objeto será bastante más complicado en los lenguajes de alto nivel que desde ensamblador. El compilador de un lenguaje de nivel alto es un programa mucho más complejo. En cuanto al proceso de compilación, una consecuencia adicional de que el lenguaje no dependa directamente del tipo de procesador es que desde un mismo código fuente se puede generar código objeto para diferentes procesadores. Por lo tanto, sólo es necesario disponer de un compilador diferente para cada tipo de procesador que se quiera soportar.

Ya que para cada archivo de código fuente se genera un fichero de código objeto, tras el proceso de compilación hay un paso adicional llamado *enlazamiento* (*link*), en el que estos dos códigos se combinan para generar un único archivo ejecutable. Coloquialmente, cuando le pedimos que compila un programa ya se suele dar por hecho que también se enlazará, en su caso. Aun así, formalmente se consideran dos pasos diferenciados.

Figura 2 Proceso de compilación (y enlazado) del código fuente



Errores de compilación

El compilador es fundamental para generar un programa en un lenguaje compilado, ya sea de nivel alto o bajo. Para poder hacer su trabajo de manera satisfactoria y generar código objeto a partir del código fuente necesario que las instrucciones sigan perfectamente la sintaxis del lenguaje elegido. Por ejemplo, hay que usar sólo las instrucciones especificadas en el lenguaje y hacerlo en el formato adecuado. Si no es así, el compilador es incapaz de entender el orden que se quiere dar al ordenador y no sabe cómo traducirla a lenguaje máquina.

Cuando el compilador detecta que una parte del código fuente no sigue las normas del lenguaje, el proceso de compilación se interrumpe y anuncia que hay un **error de compilación**.

Cuando esto ocurre, habrá que repasar el código fuente e intentar averiguar dónde está el error. Normalmente, el compilador da algún mensaje sobre el que considera que está mal.

Hay que ser conscientes de que un programador puede llegar a dedicar una buena parte del tiempo de la generación del programa en la resolución de errores de compilación, aunque hoy en día los entornos de desarrollo facilitan enormemente esta tarea.

Ahora bien, que un programa compile correctamente sólo significa que se ha escrito de acuerdo con las normas del lenguaje de programación, pero no aporta ninguna garantía de que sea correcto, es decir, que haga correctamente la tarea para la que se ha ideado.

2.2 Lenguajes interpretados

En contraposición a los lenguajes compilados, tenemos los lenguajes interpretados. Como en el caso de los lenguajes compilados, los programas también se escriben en archivos de texto que contienen código fuente. La diferencia surge inmediatamente después de terminar de escribirlos, en la forma en que se genera un archivo ejecutable. La cuestión es que, precisamente, ni se genera ningún código objeto ni ningún archivo ejecutable. Se trabaja directamente con el archivo de código fuente. Cuando el trabajo escrito, el proceso de creación del programa ejecutable ha finalizado.

Un intérprete no traduce el código fuente del programa en código objeto y entonces lo ejecuta. Lo que hace es ejecutar diferentes instrucciones de su propio código según va leyendo las instrucciones del código fuente.

El intérprete va leyendo las instrucciones del código fuente una por una y las procesando de manera que actúa de una manera o de otra, es decir, ejecuta una parte u otra de su propio código objeto según el tipo de instrucción leída. A fin de cuentas, sería un programa que imita el comportamiento de un procesador, pero a escala de software.

Un programa en un lenguaje interpretado se ejecuta indirectamente, mediante la ayuda de un programa auxiliar llamado **intérprete**, que procesa el código fuente y gestiona la ejecución.

De igual manera que en los lenguajes compilados, puede suceder que el programador haya incluido sin darse cuenta algún error de sintaxis en las instrucciones. En este caso, será el intérprete quien mostrará el error y se negará a ejecutar el programa hasta que haya sido solucionado.

En general la ejecución de programas escritos en lenguajes interpretados es más lenta en que la de los lenguajes compilados.

Por sus características, los lenguajes interpretados no requieren un proceso posterior de enlazado.

Entre los lenguajes interpretados más conocidos encontramos JavaScript, PHP o Perl. Muchos son lenguajes de *script*, que permiten el control de aplicaciones en un sistema operativo, llevar a cabo procesos por lotes (*batch*) o generar dinámicamente contenido web.

2.3 Lenguajes híbridos

Algunos lenguajes interpretados usan una aproximación híbrida. El código fuente se compila y como resultado se genera un fichero de datos binarios escrito en código. En Java este código intermedio se denomina *bytecode*, sin embargo, no es formalmente código objeto, ya que no es capaz de entenderlo el hardware de ningún procesador. Sólo un intérprete lo puede

procesar y ejecutar. Simplemente es una manera de almacenar más eficiente y en menos espacio, en formato binario y no en texto, las instrucciones incluidas en el código fuente. Este es el motivo por el que, a pesar de necesitar un proceso de compilación, estos lenguajes no se consideran realmente compilados.

Entre los lenguajes híbridos encontramos a Java o C#.

3 Entornos de desarrollo

Una vez se ha descrito el proceso general para desarrollar y llegar a ejecutar un programa, se hace evidente que hay que tener instalados y correctamente configurados dos programas completamente diferentes e independientes en su ordenador para desarrollarlos: editor, por una parte, y compilador (incluyendo el enlazador) o intérprete por la otra, según el tipo de lenguaje. Cada vez que desee modificar y probar su programa deberá ir alternando ejecuciones entre ambos. Realmente, sería mucho más cómodo si todo ello se pudiera hacer desde un único programa, que integrase los tres. Un editor avanzado desde el que se pueda compilar, enlazar en su caso, e iniciar la ejecución de código fuente para comprobar si funciona.

Un **IDE** (I*ntegrated Development Environment* o entorno de desarrollo) es una herramienta que integra todo lo necesario para generar programas de ordenador, de manera que el trabajo sea mucho más cómodo.

Algunos ejemplos de IDE son Visual Studio, para los lenguajes C #, C ++ y Visual Basic; Eclipse para Java; Netbeans, para los lenguajes Java y Ruby; Dev-Pascal, para el lenguaje Pascal, o el Dev-C, para el lenguaje C.

La utilización de estas herramientas agiliza increíblemente el trabajo del programador. Además, los IDE más modernos van más allá de integrar editor, compilador y enlazador o intérprete, y aportan otras características que hacen aún más eficiente la tarea de programar. Por ejemplo:

- Posibilidad de hacer resaltar con códigos de colores los diferentes tipos de instrucciones o aspectos relevantes de la sintaxis del lenguaje soportado, para facilitar la comprensión del código fuente.
- Acceso a documentación y ayuda contextual sobre las instrucciones y sintaxis de los lenguajes soportados.
- Detección, y en algunos casos incluso corrección, automática de errores de sintaxis en el código, de manera similar a un procesador de texto. Así, no es necesario compilar para saber que el programa está mal.
- Soporte simultáneo del desarrollo de lenguajes de programación diferentes.
- Un depurador, una herramienta muy útil que permite pausar la ejecución del programa en cualquier momento o hacerla instrucción por instrucción, por lo que permite analizar cómo funciona el programa y detectar errores.
- Sistemas de ayuda para la creación de interfaces gráficas.