**Tema 6**

**Lenguajes de programación. Características, elementos y funciones**

**en entornos Java, C, C++ y .Net.**

**1. Introducción**

**2. Conceptos básicos**

**3. Introducción a los lenguajes de programación**

**4. Generaciones en los lenguajes de programación**

**5. Tipos de programación**

**6. Procesos en la programación**

**7. Tipos de lenguajes de programación**

**8. Estilos de programación**

**9. Otros conceptos base en programación**

**10. Aplicaciones de los lenguajes de programación**

**11. Historia de los lenguajes de programación**

**12. Algunos lenguajes de programación**

**13. Otros lenguajes de programación**

**14. Características, elementos y funciones de JAVA, C y C++**

**15. Entornos de programación visual**

**16. .NET Framework**

**17. Clasificación de los lenguajes de**

**programación**

**Anexos:**

Anexo I. Lenguaje C

Anexo II. Lenguaje C++

Anexo III. Lenguaje Java

Anexo IV. Visual Basic.Net

Anexo V. C#.Net

**Guion-resumen**

**1. Introducción**

Desde que surgieron las primeras computadoras ha sido necesario definir lenguajes que le permitan al hombre comunicarse estas. Esta máquina (el ordenador) fue diseñada para realizar procesos internos en virtud de entradas y salidas de datos. En la actualidad hay equipos computacionales que son automáticos pero, aun así, necesitan comunicarse internamente de alguna forma.

El diseño de soluciones a la medida de nuestros problemas requiere una metodología que nos enseñe de manera gradual la forma de llegar a las soluciones. A las soluciones creadas por computadora se las conoce como programas y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, con un grupo de datos específicos.

Es decir, un programa nos sirve para solucionar un problema específico. Para poder realizar programas, además de conocer la metodología mencionada, también debemos conocer, de manera específica, las funciones que pueden realizar las computadoras y las formas en que se pueden manejar los elementos que hay en las mismas.

Como ya sabemos, un ordenador es un conjunto de circuitos, cables, etc. por los cuales circulan impulsos eléctricos que representan el código binario (0 y 1). Entonces, ¿cómo vamos a poder hacer que un conjunto de circuitos desempeñen una determinada tarea y nos entreguen los resultados que nosotros esperamos?, es decir, ¿de qué manera se puede lograr la comunicación entre el hombre y el ordenador? Es este el papel que juegan los lenguajes de programación.

**2. Conceptos básicos**

Un programa es una serie o secuencia de instrucciones que el ordenador debe ejecutar para realizar la tarea prevista por el programador. La CPU solo ejecuta las instrucciones que componen el programa: operaciones aritméticas, operaciones lógicas, comparaciones y movimientos de datos.

Cuando nos planteamos un problema complejo y queremos resolverlo con la utilización del ordenador, necesitamos descomponerlo en una serie de tareas simples que se irán repitiendo a lo largo de un proceso hasta la resolución del problema; el ordenador ha realizado una tarea compleja, a partir de instrucciones simples. Dicho conjunto de tareas simples sería el programa y su elaboración es lo que entendemos por programación.

La determinación de la calidad de los programas estará en función de las ventajas de su utilización; para ello existen unas características entre la cuales citamos:

— Legible. Todo programa debe ser de fácil comprensión no solo por los futuros usuarios, sino por cualquier programador.

— Flexible. Capaz de adaptarse con facilidad a los cambios que puedan producirse en el planteamiento inicial.

— Portable. Facilidad para compilarse o interpretarse en distintas máquinas y sistemas operativos, también un factor a tener en cuenta sería su facilidad para ser traducido a otros lenguajes de programación.

— Fiable. El programa debe ser capaz de recuperar el control cuando su utilización no sea la adecuada.

— Eficaz. El programa ha de utilizar eficazmente los recursos de que disponga, tanto del sistema operativo como del equipo en que trabaje.

Cuando nos disponemos a programar, la primera decisión que hemos de tomar es la elección del lenguaje a emplear, es decir, la forma en la que el programador tiene que escribir las operaciones a realizar por el ordenador.

Cada computadora tiene un solo lenguaje que puede ejecutarse: el lenguaje de máquina.

Hablamos de programar en lenguajes de alto nivel, pero estos lenguajes deben ser traducidos al lenguaje de máquina de la computadora con que estamos trabajando. Estos lenguajes de alto nivel son un medio de facilitar la labor del programador.

Los programas en lenguaje máquina (primera generación) están escritos en el nivel más básico de operación de la computadora. Las instrucciones están dirigidas a ese nivel, el lenguaje máquina y los lenguajes programadores de segunda generación, que utilizan símbolos para las instrucciones reciben la designación de lenguaje de bajo nivel. Programar en estos lenguajes resulta ser arduo y tedioso. Casi toda la programación se hace en lenguajes de alto nivel (de la tercera generación y subsiguientes).

Un lenguaje de programación es un conjunto de reglas, notaciones, símbolos y/o caracteres que permiten al programador poder expresar el pensamiento de datos y sus estructuras en la computadora, usando también una sintaxis y una gramática determinada.

Análogamente, diremos que un programa es un conjunto de órdenes o instrucciones que resuelven un problema específico basado en un lenguaje de programación.

**Los lenguajes de programación se clasifican según su base de desarrollo y su uso en:**

— Lenguajes basados en cálculo numérico: Fortran, Maple, Mathlab y Algol.

— Lenguajes para negocios: Cobol.

— Lenguajes para la inteligencia artificial: Prolog, Adal, Lisp y Logo.

— Lenguajes para **sistemas: C y ensamblador.**

También se pueden clasificar según la forma de ejecutar los programas:

— Lenguajes imperativos. Son aquellos que son controlados por instrucciones imperativas. Pascal, Fortran y otros manejan este modelo.

— Lenguajes aplicativos o funcionales. Son aquellos que manejan una preaplicación y dan una prerrespuesta antes de aplicarlo realmente.

— Lenguajes con base en reglas. Son los que ejecutan instrucciones en base al cumplimiento de ciertas condiciones.

— Lenguajes orientados a objetos. Son los que manejan muchas instrucciones por medio de un objeto y que son controladas por pocas funciones.

Generalizando, definiremos los siguientes términos:

— Lenguaje. Un conjunto de símbolos, caracteres y reglas que permiten a los programadores comunicarse con las computadoras para que realicen algo.

— Lenguaje de alto nivel. Lenguaje que se basa en instrucciones más globales y más poderosas, tal como los Visuales, C++ y otros más. El archivo resultado de un lenguaje de alto nivel es más grande que los de lenguaje de máquina.

— Lenguaje de máquina. Lenguaje que usa instrucciones más directas hacia el procesador de la computadora, las cuales son más simples y más sencillas.

**3. Introducción a los lenguajes de programación**

El lenguaje de programación es la forma en la que el programador escribe las operaciones que el ordenador debe realizar.

La CPU esta preparada para manejar unas instrucciones escritas en un tipo de lenguaje muy simple llamado lenguaje máquina. Cada modelo de CPU posee su propio lenguaje máquina y puede ejecutar un programa solo si está escrito en ese lenguaje (para poder ejecutar programas escritos en otros lenguajes, es necesario primero trasladarlos a lenguaje máquina).

El ordenador ejecuta mecánicamente los programas en lenguaje máquina, esto es, sin entenderlos o pensar sobre ellos, simplemente porque es la única forma física de hacerlo.

Las instrucciones del lenguaje máquina están expresadas con números binarios. Un número binario está compuesto únicamente por dos dígitos, cero y uno. Por tanto, las instrucciones del lenguaje máquina son una secuencia de ceros y unos. **Cada secuencia concreta indica una instrucción determinada**. Un interruptor ON, representa un uno, mientras que si está OFF, representa cero. **Las instrucciones máquina están almacenadas en la memoria como conjuntos de interruptores en ON y en OFF**. El ordenador realiza los cálculos por medio de estos interruptores que siguen un patrón determinado al ejecutar cada una de las instrucciones del programa.

En función de su parecido con el lenguaje natural, podemos hablar de lenguajes de bajo nivel y lenguajes de alto nivel

. En los primeros la sintaxis está más próxima al lenguaje máquina que al lenguaje humano y en los de alto nivel es todo lo contrario. Cuando un programa es ejecutado directamente por el ordenador, es decir está en código máquina, decimos que es un lenguaje de bajo nivel. Casi todos los programas

son escritos en lenguajes de alto nivel como Java, Pascal o C++.

|  |  |  |
| --- | --- | --- |
| Programa Fuente | Traductor | Programa Objeto |
| (Escrito por el programador) | (compilador o intérprete) | (Lenguaje máquina) |

**4. Generaciones en los lenguajes de programación**

La clasificación de los lenguajes de programación viene estipulada por su aproximación al lenguaje utilizado por el ordenador, teniendo en cuenta que todo lenguaje tiene que acabar siendo traducido al propio lenguaje del ordenador.

Según este criterio, los niveles de clasificación se pueden establecer por generaciones.

**4.1. Lenguajes de primera generación**

En este primer bloque nos vamos a encontrar con un único lenguaje denominado lenguaje máquina difícil de diseñar, ya que está en relación directa con el hardware. Es el denominado lenguaje del ordenador, formado por tan solo dos valores, el 1 y el 0, que representan los niveles alto y bajo de tensión. Así, cualquier expresión (instrucción) que se le quiera comunicar al procesador a través del programa deberá realizarse únicamente como expresión de 1 y 0, siguiendo una tabla de codificación interna de la que debe disponer

el propio procesador.

Con este lenguaje la tarea de programar se hace larga y tediosa y la longitud de los programas resultantes es muy grande, por lo que su manejo pasa a ser complicado, así como las correcciones del mismo. Por el contrario, dispone de la ventaja de ser el lenguaje que más rápidamente se interpreta, al ser precisamente el propio lenguaje del procesador.

**4.2. Lenguajes de segunda generación**

Con el paso del tiempo los programadores, cansados de escribir códigos máquina, llegaron a la conclusión de que todas aquellas expresiones que utilizaban en los programas, tarde o temprano volvían a repetirse, con lo que debían volver a rescribir el mismo código.

Esto les llevó a idear una serie de abreviaturas denominadas nemotécnicos, consistentes en la representación de códigos binarios.

Gracias a este invento cada programador tenía la facilidad de resumir todos sus códigos en muchas menos líneas, ya que las expresiones que necesitaban ser repetidas se hacían de forma abreviada gracias a estos nemotécnicos.

Este lenguaje resultante recibió el nombre de Ensamblador. Y podemos destacar de él:

— Su menor tamaño de programa con respecto al lenguaje máquina.

— Su menor velocidad de ejecución, porque por supuesto todos los nemotécnicos deben ser traducidos por el procesador, invirtiendo un tiempo en ello que los hace más lentos que los lenguajes máquina.

— Su dependencia todavía del hardware.

— Su menor complejidad a la hora de ser diseñado, ya que cada programador conocía perfectamente sus propios nemotécnicos.

A los lenguajes de primera y segunda generación se les conoce con el nombre de Lenguajes de Bajo Nivel, ya que son próximos al lenguaje del ordenador.

**4.3. Lenguajes de tercera generación**

Para reducir las deficiencias de los lenguajes de primera y segunda generación, surgieron nuevos lenguajes denominados de tercera generación, que se acercaban al lenguaje humano y por tanto se alejaban del lenguaje máquina. Entre sus propiedades cabe destacar:

**— Las instrucciones se obtienen mediante expresiones que tienen algún significado para el lenguaje humano**: While, if, End, Then, etc. (Mientras, Si, Fin, Entonces,

etc.).

— Los programas sufren una reducción a la mínima expresión.

— La construcción por parte del programador es más sencilla.

— Son independientes del hardware, aunque todavía se tiene problemas con respecto al sistema operativo en el que se trabaja.

— La corrección de códigos es sencilla.

— Su velocidad de ejecución disminuye con respecto a los lenguajes anteriores, pero con los avances tecnológicos, este dato parece carecer de importancia.

A los lenguajes de tercera generación se les dio el nombre de Lenguajes de Alto Nivel precisamente por su lejanía del lenguaje máquina.

Estos lenguajes los podemos clasificar a su vez según un criterio de utilización:

**— Lenguajes de propósito general**: son aquellos que son utilizados para la resolución de todo tipo de problemas, como por ejemplo **C1, Cobol, Basic**.

**— Lenguajes de propósito específico:** son aquellos que son utilizados para resolver problemas determinados según un área específica, como, por ejemplo, **FORTRAN** para resolver cálculos numéricos, **COBOL** para resolver problemas relacionados con la gestión de empresas, como contabilidad, nóminas, etc.

**4.4. Lenguajes de cuarta generación**

Con la aparición de estos lenguajes, el programador prácticamente se desentiende de “programar”, ya que sus expresiones son prácticamente similares al lenguaje humano.

Los podemos clasificar en:

**• Generadores de código.** Genera de forma automática o semiautomática programas en lenguaje de alto nivel, empleando un número mucho menor de instrucciones que las que se necesitarían con los propios lenguajes de alto nivel.

**• Petición.** Son la mayoría de ellos, y se basan en el manejo de bases de datos.

**5. Tipos de programación**

Para la obtención de programas que reúnan los máximos requisitos de calidad, la programación ha ido evolucionando al mismo tiempo que los métodos y técnicas. Se han ido estableciendo diferentes criterios de programación: secuencial, modular, estructurada, orientada a objetos; son complementarias entre sí y por ello no excluyentes para la obtención de los programas requeridos.

**5.1. Programación secuencial**

En los primeros pasos de los lenguajes de programación, estos es escribían de arriba a abajo, todo ello en un solo archivo sin el uso de funciones ni modularidad. Si un trozo de código tenía un error había que revisarlo todo y volver a compilar el programa entero.

**5.2. Programación modular**

En la programación modular, descomponemos el problema en una serie de operaciones simples cuya realización nos lleva a la resolución del problema complejo planteado; ahora bien, esta descomposición en operaciones simples la agruparemos en módulos que funcionen independientes entre sí y que sean independientes del programa en el que actúen.

En la programación modular encontraremos un programa que será el **programa principal o módulo raíz.** Dicho módulo **estará compuesto principalmente de llamadas a otros módulos secundarios**.

La programación modular **es una programación lineal**, sencilla, donde aparece **un punto de entrada** o de comienzo y **se va desarrollando de arriba a abajo (top-down)** con **un punto final** o de salida.

**5.3. Programación estructurada**

La programación estructurada nos permite resolver problemas a partir de un solo punto de entrada (inicio) y otro de salida (final); su estructura con una composición lineal permite la utilización de estructuras más complejas:

— Estructura secuencial: operaciones consecutivas.

— Estructura condicional: selección entre dos o más módulos si se cumple x.

— Estructura repetitiva: se repiten una o varias veces dependiendo de una condición expresa.

**5.4. Programación orientada a objetos**

Ante la dificultad para la optimización de los programas y la creciente complejidadde las aplicaciones, *partiendo de la programación estructurada*, a la que engloba, y dotando al programador de nuevos elementos para el análisis y desarrollo de software comienza la programación orientada a objetos, facilitando la producción de sistemas cada vez más complejos, que **permiten modelar problemas no estructurados**, incrementan la productividad gracias a la **reutilización de objetos** y **facilitan su mantenimiento**.

En la programación orientada a objetos (POO) ***se utilizan conceptos y herramientas que modelan y representa el mundo real tan fielmente como sea posible***.

La POO proporciona ventajas sobre otros lenguajes de programación:

**— Uniformidad**. La representación de los objetos lleva implícita tanto el análisis como el diseño y codificación de los mismos.

**— Compresión.** Los datos que componen *los objetos y los procedimientos* que los manipulan, *están agrupados en clases,* que se corresponden con las estructuras de información que el programa trata.

**— Flexibilidad.** Al relacionar procedimientos y datos, *cualquier cambio* que se realice sobre ellos *quedará reflejado automáticamente* en cualquier lugar donde estos datos aparezcan.

**— Reutilización.** La noción de objeto permite que programas que traten las mismas estructuras de datos reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, *el desarrollo de un programa puede llegar a ser una simple combinación*

*de objetos ya definidos* donde estos están relacionados de una manera particular.

La POO no sustituye a ninguna metodología ni lenguaje de programación anterior.

Todos los programas que se realizan según POO se pueden realizar igualmente mediante programación estructurada. Su uso en la actualidad se justifica porque el desarrollo de todas las nuevas herramientas basadas en una interface de usuario gráfico como Windows, X- Windows, etc. es mucho más sencillo.

La POO fue estudiada en detalle en el Tema 5.

**6. Procesos en la programación**

Fases a seguir en todo proceso de programación:

**1. Análisis detallado del problema a resolver**

Plantearemos el problema que deseamos resolver, analizándolo detalladamente; para ello tendremos en cuenta una serie de factores como son:

**— El equipo con el que contamos (hardware y software**). Los datos iniciales o datos de entrada.

— **Tratamiento u operaciones que realizaremos** con esos datos de entrada.

**— Los resultados o datos** de salida que queramos obtener.

En este análisis estudiaremos las posibilidades de **descomposición del problema en módulos** más simples que faciliten la tarea de programación y de ejecución.

2. Diseño del algoritmo

Este diseño del algoritmo sería la descomposición de la ejecución del problema en tareas elementales. El algoritmo describirá la realización del problema complejo en operaciones básicas, sencillas y elementales a realizar por el ordenador.

Estas operaciones son las instrucciones u órdenes que podemos dar al ordenador y se corresponden con las diferentes estructuras de control utilizadas en la programación estructurada; en función de dichas estructuras, se pueden agrupar en las siguientes instrucciones:

**— Secuencias**. Son instrucciones que se irán realizando ininterrumpidamente y en el orden exacto en el que estén transcritas.

Primera generación FORTRAN

Segunda generación ALGOL-60

Tercera generación Pascal

Cuarta generación Ada

Programación orientada a objetos Smalltalk y C++

Programación funcional LISP, Scheme y ML

Programación orientada a la lógica PROLOG

**— Iteraciones (bucles).** Son conjuntos de instrucciones que se ejecutarán repetidamente mientras se cumplan unas condiciones determinadas.

**— Decisiones.** El ordenador tomará ciertas decisiones, en función de los resultados que se produzcan en el transcurso del programa, escogiendo diferentes itinerarios u opciones diferentes.

**— Saltos.** El ordenador desviará la secuencia lineal de instrucciones, saltando de unas a otras instrucciones en función de ciertas condiciones.

3. Programación

Es la codificación del algoritmo o programación propiamente dicha en el lenguaje de programación elegido. El programador, en función de dicho lenguaje, utilizando la sintaxis y el vocabulario requerido, irá traduciendo el algoritmo al lenguaje de programación, utilizando las funciones que se corresponderán con las diferentes tareas a ejecutar, estructuras, etc.

4. Obtención del programa

Una vez codificado el algoritmo, necesitamos **obtener el programa ejecutable**, es decir el software que resolverá el problema de partida. Esta tarea se resolverá en tres fases: **edición, compilación y montaje o linkado**.

La edición del programa es la escritura del mismo en el lenguaje de programación y dentro del sistema operativo utilizado; obtendremos el programa en un **fichero de texto o código fuente.**

La compilación es la traducción del fichero de texto en lenguaje máquina, obteniéndose los ficheros código objeto o código máquina. Dicho proceso lo realizarán los **programas compiladores o intérpretes.**

**El montaje o linkado es el proceso** de unión entre las distintas partes del lenguaje máquina para la obtención del fichero o programa ejecutable.

5. Depuración del programa

La depuración del programa es la comprobación de la bondad del mismo; estudiaremos

el funcionamiento del programa en todas las situaciones posibles comprobando

su correcto funcionamiento, para ello iremos introduciendo los diferentes

valores posibles y comprobando el resultado; tendremos que volver al

fichero de texto o código fuente cuantas veces sea necesario hasta que el definitivo

programa ejecutable cumpla todas las condiciones requeridas, en ese

momento el resultado es el óptimo y el programa está terminado.

6. Documentación del programa

Es la elaboración de la documentación técnica del programa. Se realizará

en función del programa y de su futura utilización o comercialización.

Dicha documentación se realizará a dos niveles:

6-11

Lenguajes de programación

• Documentación interna. Se elaborará una documentación a nivel de programación,

especificándose cuantas aclaraciones y comentarios sean necesarios.

Podrán ir dentro del propio programa y su función es simplificar la

actualización del programa.

• Documentación externa. La documentación será a nivel de usuario, en la

que se especificará con toda claridad la instalación del programa, las condiciones

técnicas o el sistema requerido para su funcionamiento, así como la

descripción del funcionamiento del programa, datos de entrada, salida, etc.

Los principales errores en la ejecución de un programa son:

— Datos de entrada incorrectos que producen una parada del sistema (por ejemplo,

introducir un dividendo con valor cero en una operación de división).

— Bucles mal definidos que producen un funcionamiento continuo del programa

(por ejemplo, un bucle sin fin o bucle infinito).

— Datos de salida incorrectos, producidos por un mal desarrollo del programa o

ambigüedad en las especificaciones del usuario.

**7. Tipos de lenguajes de programación**

Dentro de los lenguajes de alto nivel podemos distinguir entre lenguajes compilados y lenguajes interpretados; dicha distinción se realiza en función de la forma en que generamos el programa ejecutable. Un lenguaje de alto nivel no puede ser ejecutado directamente por ningún ordenador; es necesario trasladarlo a lenguaje máquina.

**7.1. Lenguajes compilados**

Los lenguajes compilados son aquellos que, una vez escrito el código fuente, permite obtener un programa ejecutable y autónomo, susceptible de ser ejecutados en cualquier máquina bajo el sistema operativo para el que fue diseñado. En estos lenguajes de programación es imprescindible la existencia de un compilador***. El compilador es un programa complejo que traduce todo el programa de golpe en código máquina***. Es decir, el programa compilador traduce las instrucciones de un lenguaje de alto nivel en instrucciones de lenguaje máquina que la computadora puede interpretar y ejecutar. Un compilador traduce en instrucciones de lenguaje de máquina las instrucciones de lenguaje de alto nivel, llamadas programa fuente. El resultado de la compilación es el programa objeto*. El código objeto también se suele denominar código binario o código máquina.* Una vez que el programa está compilado, las líneas de código fuente dejan de tener sentido durante la ejecución del programa.

**7.2. Lenguajes interpretados**

Los lenguajes interpretados son aquellos en los que el programa fuente necesita de un programa intérprete para ser ejecutado.

*El intérprete traduce instrucción por instrucción a medida que va siendo necesario.* Un intérprete es un programa que trabaja de una forma muy semejante a la CPU, con una especie de ciclo de leer y ejecutar. *Para poder ejecutar un programa, el intérprete ejecuta un bucle en el que va leyendo las instrucciones una a una, decide si se han de ejecutar y, si es así, las convierte en el código máquina apropiado.* El intérprete puede permitir el uso de un programa en código máquina de un ordenador, en otro tipo de ordenador completamente diferente. Un intérprete cumple las mismas funciones que un compilador, aunque en diferente forma. En vez de traducir íntegramente el programa fuente en una sola pasada, traduce y ejecuta cada instrucción antes de pasar a la siguiente. La ventaja de los intérpretes sobre los compiladores radica en que, si hay error en la sintaxis de instrucciones, este se indica al programador de inmediato, con lo cual se le permite hacer las correcciones durante el desarrollo del programa*. Las desventajas del intérprete consisten en que no utiliza los recursos de la compilación con la misma eficiencia con que un programa que ha sido compilado.* Como el intérprete no produce programa objeto, debe hacer el proceso de traducción cada vez que un programa se mueve, línea por línea. Con la compilación separada, el programa, por la extensión del código fuente, no puede ser compilado en un solo bloque, con lo que se compilará por partes, obteniéndose diferentes códigos objetos; una vez completado todo el programa, todos los códigos objetos

que conforman un programa se agruparán obteniéndose el código o fichero ejecutable.

Los lenguajes ensambladores y máquina son dependientes de la máquina.

***Cada tipo de máquina tiene su propio lenguaje máquina distinto y su lenguaje ensamblador asociado.***

El lenguaje ensamblador es simplemente una representación simbólica del lenguaje máquina asociado, lo cual permite una programación menos tediosa que con el anterior.

La programación en un lenguaje de alto nivel o en un lenguaje ensamblador requiere algún tipo de interfaz con el lenguaje máquina para que el programa pueda ejecutarse.

***Las tres interfaces más comunes son: un ensamblador, un compilador y un intérprete.***

El ensamblador y el compilador traducen el programa a otro equivalente en el lenguaje de la máquina residente como un paso separado antes de la ejecución.

Por otra parte, el intérprete ejecuta directamente las instrucciones en un lenguaje de alto nivel, sin un paso de procesamiento previo.

***La compilación es un proceso más eficiente que la interpretación para la mayoría de los tipos de máquinas.*** Esto se debe principalmente a que las sentencias dentro de un bucle deben ser interpretadas cada vez que se ejecutan por un intérprete. Con un compilador, cada sentencia es interpretada y luego traducida a lenguaje máquina solo una vez.

Algunos lenguajes son principalmente interpretados, como APL, PROLOG y LISP, JAVA, etc.

Ejemplos de lenguajes compilados son: PASCAL, FORTRAN, COBOL, PL/I, SNOBOL, C, ADA, etc.

En algunos casos, un compilador estará utilizable alternativamente para un lenguaje interpretado y viceversa.

**8. Estilos de programación**

De acuerdo con el estilo de programación, podemos clasificar los lenguajes en las siguientes categorías:

**• Imperativos:** son aquellos lenguajes que basan su funcionamiento en un conjunto de **instrucciones secuenciales**, las cuales, al ejecutarse, van alterando las regiones de memoria donde residen todos los valores de las variables involucradas en el problema que se plantea resolver. Es decir, ***se cambia progresivamente el estado del sistema, hasta alcanzar la solución del problema.***

***Está basada en el modelo Von Neumann***, en donde un conjunto de operaciones primitivas realizan una ejecución secuencial. Realiza una abstracción en el manejo de variables, expresiones e instrucciones y para programar es necesario declarar las variables necesarias y diseñar una secuencia adecuada de instrucciones

(asignaciones). Algunos de los lenguajes de este tipo ***son Pascal, Ada y C.***

**• Declarativos:** en este paradigma, más que el cómo desarrollar paso a paso un proceso, nos interesa el qué deseamos obtener a través del programa**. El ejemplo típico de lenguaje declarativo es SQL,** el cual es utilizado para interactuar con la información de bases de datos, concentrándose solo en los resultados que van a ser obtenidos, dejándole al traductor la tarea de cómo llegar a ellos y presentárnoslos. Dentro de este paradigma, se encuentran dos estilos distintos de programación, cada uno de los cuales posee su propia lógica:

***— Funcionales:*** son lenguajes basados en funciones, las cuales se representan mediante expresiones que nos permiten obtener ciertos resultados a partir de una serie de argumentos.

De hecho, las expresiones están formadas por un conjunto de términos, que a su vez pueden encapsular otras expresiones, para que con la evaluación de todas ellas, llegar a la solución deseada. El programa es una función (o un grupo de funciones). La relación entre las funciones es muy simple: una función puede llamar a otra función, o el resultado de una función puede ser usado como el argumento de otra función. Las variables, comandos y efectos laterales son exclusivos. Los programas son escritos enteramente dentro del lenguaje de expresiones, funciones y declaraciones. Dos de estos lenguajes son Scheme y ML.

***— Lógicos:*** este tipo de lenguajes se basan en el cálculo de predicados, la cual es una teoría matemática que permite, entre otras cosas, lograr que un ordenador, basándose en un conjunto de hechos y de reglas lógicas, pueda derivar en soluciones inteligentes. La programación lógica está basada en la noción de relación, debido a que la relación es un concepto general de una aplicación. La programación lógica es potencialmente de alto nivel. Los lenguajes de programación lógica pueden explotar la inteligencia artificial. Un lenguaje de este tipo **es Prolog.**

**• Programación orientada a objetos:** los programas de este tipo, se concentran en los objetos que van a manipular y no en la lógica requerida para manipularlos. La programación orientada al objeto está basada en los objetos, clase, método, envío y recepción de mensajes, herencia y polimorfismo. Algunos de los lenguajes

de este tipo son C++, JAVAy Smalltalk.

La orientación a objetos se puede definir como una disciplina de ingeniería de desarrollo y modelado de software que permite construir más fácilmente sistemas complejos a partir de componentes individuales.

Permite una representación más directa del modelo del mundo real, reduciendo fuertemente la transformación radical normal desde los requerimientos del sistema, definidos en términos del usuario, a las especificaciones del sistema, definidasen términos del computador.

Actualmente la tendencia de la ingeniería informática es producir componentes reutilizables para ensamblarlos unos a otros y obtener así el producto completo. Estos elementos reutilizables son denominados “**componentes integrados de software” (CIS)** por su teórica similitud con los **“componentes integrados de hardware” (chips),** innovación que revolucionó la industria del computador en los años 70. El paradigma orientado a objetos es, pues, una filosofía de desarrollo y empaquetamiento de software que permite crear unidades funcionales extensibles y genéricas, de forma que el usuario las pueda aplicar según sus necesidades y de acuerdo con las especificaciones del sistema a desarrollar.

**La orientación a objetos proporciona mejores herramientas para:**

— **Modelar el mundo real** de un modo más cercano a la perspectiva del usuario.

**— Interactuar** fácilmente con un entorno computacional, usando metáforas familiares.

**— Construir componentes** reutilizables de software y bibliotecas específicas de estos componentes fácilmente extensibles.

**— Modificar y ampliar** con facilidad la implementación de estos componentes sin afectar al resto de la estructura.

En cuanto a los elementos fundamentales que configuran el paradigma orientado a objetos, se suelen significar siete:

**— Estructura modular basada en objetos**, dado que los sistemas en esta metodología son modularizados sobre la base de sus estructuras de datos.

**— Abstracción de datos,** porque los objetos son descritos como implementaciones de tipos abstractos de datos.

**— Gestión automática de memoria**, de forma que los objetos no utilizados sean desasignados por el propio sistema sin intervención del programador.

**— Clases,** en las que cada tipo no simple sea un módulo, y cada módulo de alto nivel sea un tipo.

**— Herencia,** que permita que una clase sea definida como una extensión o restricciónde otra.

**— Polimorfismo y enlace dinámico**, de forma que las entidades del programa puedan referenciar en tiempo de ejecución a objetos de diferentes clases.

**— Herencia múltiple y repetida** para que se pueda declarar una clase como heredera de varias, e incluso de ella misma.

El paradigma orientado a **objetos se describe a menudo usando el concepto de objeto/mensaje**, en el ***que cada objeto (elemento autónomo de***  (Lenguaje máquina)

***información creado en tiempo de ejecución) es solicitado para realizar un determinado servicio mediante el envío a ese objeto del mensaje apropiado***. El solicitante no precisa conocer cómo el objeto proporciona el servicio pedido; la implementación es interna al objeto y la gestiona el suministrador del mismo. El énfasis se produce en qué se puede obtener más que en cómo se obtiene.

Un programa orientado a objetos viene definido por la ecuación:

**OBJETOS + MENSAJES = PROGRAMA**

*Aquí el objeto es una instancia de una clase*, la cual implementa un **tipo abstracto de dato (TAD).**

Y el mensaje es la información específica que se envía al objeto para que ejecute una determinada tarea.

Un TAD define conjuntos encapsulados de objetos similares, con una colección asociada de operaciones; y especifica la estructura y el comportamiento de los objetos. Las especificaciones estructuradas del TAD describen las características de los objetos pertenecientes a ese TAD, y las especificaciones de comportamiento describen qué mensajes son aplicables a cada objeto.

**• Programación orientada al evento:** esta programación es el resultado de la programación orientada al objeto. Este tipo de programación permite trabajar con objetos y clases estándar previamente definidas por la aplicación, las cuales manejan los conceptos de encapsulasión. Las herramientas que trabajan de esta

forma, por lo general, trabajan con código original de lenguajes imperativos.

*Algunas herramientas de este tipo son Visual Basic (Basic), Delphi (Pascal) y Power Builder (C).*

**8.1. Programación estructurada**

Existen en la actualidad dos formas o metodologías básicas de construcción de software:

la programación estructurada u orientada al flujo de datos, y la programación orientada

a objetos.

Es evidente que un programa tiene como finalidad la resolución de un determinado

problema, o la realización de determinada tarea, pero para ello no hay una forma única. Se

pueden hacer diferentes programas, o algoritmos de resolución, que cumplan todos ellos

un objetivo propuesto. Por esto debemos conocer las reglas o principios que nos permitan

la elección del más adecuado, pues todos ellos incidirán directamente en el coste de su

diseño y posterior mantenimiento, no olvidando nunca una premisa fundamental: se desarrolla

para mantener.

Con independencia de la metodología empleada en su construcción, los programas

deben cumplir unas características generales como: ser legibles (fáciles de leer y comprender,

por lo que hay que comentarlos ampliamente en sus partes complejas), portables

(fáciles de codificar en otros lenguajes o en otros sistemas y configuraciones físicas), fácilmente

modificables (para facilitar su mantenimiento), eficientes (para aprovechar bien los

recursos), modularizables (descomponer el problema general de arriba abajo, top-down,

en bloques o módulos a diferentes niveles) y estructurados (siguiendo un método y unas

normas básicas). Todas estas características van dirigidas tanto a facilitar su implementación

como su verificación y depuración, así como su posterior y seguro mantenimiento.

Todo ello tiene como resultado final el que los costes y el esfuerzo personal de todo el proceso

sean menores.

Si consideramos cualquier centro de desarrollo de programas con proyectos en curso,

veremos que, frecuentemente, los programadores que comenzaron el proyecto no siguen

en el mismo centro o que han pasado a trabajar en otros proyectos. Por ello es de vital

importancia que un programa desarrollado inicialmente por una persona sea fácilmente

ampliado y modificado por otra distinta. Esta es una de las ventajas de la programación

estructurada. Los programas escritos sin un determinado método suelen tener problemas

como los siguientes:

— Suelen ser demasiado rígidos, con problemas al adaptarlos a distintos

entornos y configuraciones.

6-17

Lenguajes de programación

— Los programadores pasan la mayoría del tiempo corrigiendo sus errores.

— Los programadores rehúsan el uso de programas y módulos ya escritos y en funcionamiento,

pues prefieren escribir los suyos. La comunicación entre ellos es

difícil.

— Un proyecto de varios programadores suele tener varios conjuntos diferentes de

objetivos.

— Cada programador tiene sus propios programas y esta relación se hace inseparable.

— Las modificaciones en aplicaciones y programas son muy difíciles de hacer, implican

mucho tiempo y un elevado coste de mantenimiento. Ello conduce, bien a

colocar “parches” que complican cada vez más el diseño inicial, o bien a que el

programa caiga en desuso y que frente al elevado coste de actualización se opte

por crear una nueva aplicación que sustituya a la existente.

— Deficiencias en la documentación: incompleta o no actualizada.

Se hace preciso realizar programas siguiendo técnicas o métodos estandarizados que

consiguen las características anteriormente descritas, rápida y eficazmente. Las técnicas de

programación más empleadas que permiten seguir una metodología de la programación

son la programación modular y la programación estructurada.

Estas dos técnicas suelen ser complementarias, ya que en el análisis de un problema

pueden utilizarse criterios de programación modular para dividirlo en partes independientes

y utilizar métodos estructurados en la programación de cada módulo. Por ello no debe

causarnos extrañeza que en la actualidad se difundan con gran fuerza las técnicas de programación

estructurada, cuyo objetivo principal consiste en:

— Facilitar la comprensión del programa.

— Permitir rápidamente el mantenimiento del mismo, a lo largo de su vida útil.

Una forma de simplificar el diseño de algoritmos es utilizar la técnica de diseño descendente

de programas, Top-down (de arriba abajo), que consiste en descomponer un

problema en una serie de niveles o pasos sucesivos de refinamiento (stepwise). La metodología

descendente consiste en efectuar una relación entre las sucesivas etapas de estructuración

de modo que se relacionen unas con otras mediante entradas y salidas de información.

Es decir, se descompone el problema en etapas o estructuras jerárquicas, de modo

que se puede considerar cada estructura desde dos puntos de vista: lo que hace y como lo

hace. La programación estructurada sigue completamente las directrices top-down.

En la programación tradicional se utilizan de un modo excesivo, indiscriminado, y a

veces caprichoso, las instrucciones de bifurcación condicional e incondicional,

lo que hace difícil el seguimiento de la lógica del programa y consecuentemente

sus necesarias modificaciones futuras. La programación estructurada

tiene como uno de sus fines la eliminación de los problemas descritos; por

6-18

Desarrollo de Sistemas

ello las instrucciones de bifurcación o saltos han sido eliminadas, o por lo menos seriamente

restringidas en su utilización. Es por ello que un programa estructurado, en una

secuencia normal de lectura, puede ser fácilmente leído en su totalidad, sin saltos ni búsquedas

incontroladas.

En cierto sentido, la programación estructurada ha sido precursora del diseño orientado

a objetos, dado que los programadores en programación estructurada, dentro de la

fase de diseño deben realizar sus diferentes tareas en forma de módulos, subrutinas o bloques,

los cuales son susceptibles de estandarizarse, por lo que se pueden incluir como elementos

dentro de bibliotecas de programas para su futura utilización en otros programas

e incluso para diversas aplicaciones. Así tenemos la reutilización del código. El concepto de

objeto, si bien es más amplio, puede tener como origen estos elementos de biblioteca.

Vemos pues que con la programación estructurada se consigue hoy en día producir

buen código, dado que se utilizan estructuras estándar de control para mejorar la calidad

y el mantenimiento de los programas. Las estructuras de control fomentan el desarrollo de

programas de alto nivel por expansión ordenada de bloques de programa; dado además

que las estructuras de control son limitadas, se minimiza la complejidad de los problemas

y por consiguiente se reducen los errores. Los diseñadores especifican funciones de alto

nivel con un bloque de programa, y este bloque es entonces expandido en más componentes

detallados, basándose en que la programación estructurada se auxilia en los denominados

recursos abstractos, en lugar de los recursos concretos de que se dispone en un

determinado lenguaje de programación. Así, descomponer un programa en términos de

recursos abstractos consiste en descomponer una determinada acción compleja en términos

de un número de acciones más simples, que uno es capaz de ejecutarlas o que constituyen

instrucciones disponibles de un computador.

Con todo ello, la documentación que se produce es mucho más legible. Cada bloque

debe desarrollar una función bien definida, siendo una buena práctica de programación el

intercalar comentarios interactivos que mejoren aún más su legibilidad. Se debe pues definir

cada bloque de la estructura de control y describir sus propósitos. Los enlaces con las

descripciones de salida del código también deben ser incluidos en la documentación.

Es relativamente fácil utilizar las especificaciones para crear código bien estructurado.

Las especificaciones de proceso usan palabras clave muy similares a las construcciones utilizadas

en programación estructurada. Las sentencias aritméticas o de transformación de las

especificaciones del proceso se reemplazan por la gramática utilizada en el lenguaje de programación.

Tenemos, en consecuencia, que un programa estructurado es: fácil de comprender en

su lectura; fácil de codificar en diversos lenguajes; fácil de implantar en diferentes sistemas;

fácil de documentar; fácil de mantener; eficiente; modularizable, pues es un valor añadido

por la propia técnica de diseño.

Como resumen podemos concluir que la programación estructurada es

el conjunto de técnicas que incorporan: diseño descendente (top-down),

recursos abstractos y estructuras básicas.

6-19

Lenguajes de programación

En 1966 Böhm y Jacopini demostraron que todo programa propio, sea cual sea el trabajo

que tenga que realizar, se puede hacer utilizando tres únicas estructuras de control,

que son la secuencial, la selectiva y la repetitiva. Un programa se define como propio si

cumple las tres siguientes características:

— Posee solo un punto de entrada y uno de salida o fin para control de programa.

— Existen caminos desde la entrada hasta la salida que se pueden seguir y que pasan

por todas las partes del programa.

— Todas las instrucciones son ejecutables y no existen lazos o bucles infinitos.

Así pues podemos definir la programación estructurada como aquella que utiliza

siempre una estructura con un único punto de entrada y un único punto de salida, y que

utiliza solo tres estructuras de control: la secuencial, la selectiva o alternativa (simple,

doble, múltiple), y la repetitiva:

— Estructura secuencial: se trata de una estructura con solo un punto de entrada y

uno de salida, compuesta por una serie de tareas que también tienen un solo

punto de entrada y de salida, y donde cada tarea sigue a otra en secuencia. Las

tareas se suceden de modo que la salida de una es la entrada de la siguiente, y

así sucesivamente hasta el final del proceso.

— Estructura selectiva o alternativa: se utiliza para tomar decisiones lógicas. En ella

se evalúa una condición, y en función del resultado de la misma se realiza una

opción u otra. Las estructuras selectivas pueden ser simples, dobles o múltiples.

La alternativa simple es la típica “si-entonces” (IF-THEN), donde se evalúa la

condición, y si esta es verdadera se ejecuta una determinada acción, y si es falsa

entonces no se hace nada.

La alternativa doble ejecuta una acción diferente en cada caso posible de evaluación

de la condición (verdadero o falso). Es el “si-entonces-sino” (IF-THEN-ELSE).

La alternativa múltiple se podría realizar con las dos anteriores estructuras anidadas

o en cascada, pero la legibilidad del programa podría verse comprometida.

Por ello esta estructura múltiple se incluye también. Es la típica instrucción

“según-sea, caso de” (CASE OF).

— Estructura repetitiva: es el algoritmo necesario para repetir una o varias acciones

un número determinado de veces. Estas estructuras se denominan bucles. Para

limitar el número de veces que debe repetirse el bucle hay que contar con una

condición, para lo que se suele utilizar una variable que se incrementa con cada

ejecución. Es la típica construcción “mientras” (WHILE).

6-20

Desarrollo de Sistemas

**9. Otros conceptos base en programación**

Existen algunos términos añadidos que debemos de tener claros como son:

— Código fuente. Es el texto de un programa que un usuario puede leer, normalmente

considerado como el programa en sí. El código fuente es la entrada al

compilador o intérprete.

— Código objeto. Es la traducción a través del compilador del código fuente a código

máquina, que es el que el ordenador puede leer y ejecutar. El código objeto

es la entrada al enlazador.

— Enlazador. Es un programa que enlaza módulos compilados por separado para

producir un solo programa. La salida del enlazador es un programa ejecutable.

— Tiempo de compilación. Es el tiempo que tarda el compilador en traducir el código

fuente a código objeto.

**10. Aplicaciones de los lenguajes de programación**

Las aplicaciones de los lenguajes de programación vienen dadas por el programa que

se crea con él. Los programas se pueden clasificar por diversos tipos: aquí realizaremos una

pequeña clasificación funcional, como la siguiente:

6-21

Lenguajes de programación

**APLICACIÓN LENGUAJE**

NEGOCIOS COBOL, C, 4GL, PL/I

CIENTÍFICA FORTRAN, C, C++, BASIC, PASCAL, APL, ALGOL

SISTEMAS JOVIAL C, C++, PASCAL, ADA, BASCI, MODULA

IA LISP, PROLOG, SNOBOL

EDICIÓN TEX, POSTSCRIPT

PROCESO SHELL DE UNIX, TCL; PERL, MARVEL

NUEVOS PARADIGMAS ML, SMALLTALK, EIFFEL

INTERNET (USUARIO) HTML, DHTML, XML, SCRIPT

INTERNET (SERVIDOR) PHP, ASP, JSP

**11. Historia de los lenguajes de programación**

La historia de los lenguajes de programación se remonta a la época anterior a la II

Guerra Mundial; no obstante es a partir de los años setenta cuando tiene su mayor auge.

A partir de los años sesenta, empiezan a surgir diferentes lenguajes de programación, atendiendo

a diversos enfoques, características y propósitos. Actualmente existen alrededor de

2000 lenguajes de programación y continuamente están apareciendo otros nuevos, que

prometen hacer mejor uso de los recursos computacionales y facilitar el trabajo de los programadores.

A continuación detallamos algunas referencias interesantes.

6-22

Desarrollo de Sistemas

**AÑO LENGUAJE INVENTOR USO**

1946 Plankalkul Konrad Zuse Jugar al ajedrez.

1949 Short Code Lenguaje traducido a mano.

1950 ASM (ensamblador) Lenguaje ensamblador.

1951 A-0 Grace Hopper Primer compilador.

1952 AUTOCODE Alick E. Glennie Compilador rudimentario.

1956 FORTRAN IBM Traducción de fórmulas matemáticas.

1956 COBOL Compilador.

1958 ALGOL 58

1960 LISP Interprete orientado a la Inteligencia Artificial.

1961 FORTRAN IV IBM Traducción de fórmulas matemáticas.

1961 COBOL 61 Extendido

1960 ALGOL 60 Revisado

1964 PASCAL Niklaus Wirth Programación estructurada.

1964 BASIC Universidad de Dartmouth

1965 SNOBOL

1965 APL

1965 COBOL 65

1966 PL/I

1966 FORTRAN 66 IBM

1967 SIMULA 67

1968 ALGOL 68

1968 SNOBOL4

+1970 GW-BASIC Antiguo BASIC.

1970 APL/360

1972 SMALLTALK Xerox Pequeño y rápido.

1972 C Laboratorios Bell Lenguaje con tipos.

1974 COBOL 74

1975 PL /I Lenguaje sencillo.

1977 FORTRAN 77 IBM

6-23

Lenguajes de programación

**AÑO LENGUAJE INVENTOR USO**

+1980 SMALLTALK/V Digitalk Pequeño y rápido.

1981 PROLOG Ministerio Japonés Lenguaje para la Inteligencia Artificial.

1982 ADA Ministerio Defensa EE.UU Lenguaje muy seguro.

1984 C++ AT&T Bell Laboratories PROG. ORIENTADA A OBJETOS.

1985 CLIPPER Compilador para bases de datos.

1985 QuickBASIC 1.0 Microsoft® Vompilador de BASIC.

1986 QuickBASIC 2.0 Microsoft® Soporte de tarjeta gráfica EGA.

1987 QuickBASIC 3.0 Microsoft® 43 líneas con la tarjeta EGA.

1987 QuickBASIC 4.0 Microsoft® Tarjetas Hércules, VGA.

1987 CLIPPER '87 Compilador para bases de datos.

1988 QuickBASIC 4.5 Microsoft® Tarjeta SVGA.

1989 QuickBASIC 7.1 Microsoft®

1989 ASIC V5.0 Interprete tipo QBASIC shareware.

+1990 VISUAL C++ Entorno visual de C++.

+1990 JavaScript Lenguaje de Script (GUIONES).

+1990 VBScript Microsoft® Lenguaje de Script (GUIONES).

1993 HTML Tim Berners-Lee Surge para su uso en Internet.

1993 XML C. M. Sperberg-McQueen Surge para su uso en Internet.

+1990 SGML Charles F. Goldfarb Surge para su uso en Internet.

+1990 WML Surge para su uso en Internet.

+1990 ASP Microsoft® Uso en Internet (SERVIDOR).

+1990 PHP Uso en Internet (SERVIDOR).

1995 JAVA Sun Microsystems Applets y aplicaciones.

1995 DELPHI

1995 CLIPPER 5.01 Compilador para bases de datos.

1995 GNAT ADA95 Ministerio Defensa EE.UU Lenguaje muy seguro.

1995 FORTRAN 95 IBM

1991 VISUAL BASIC 1.0 Microsoft® Entorno visual de BASIC.

1992 VISUAL BASIC 2.0 Microsoft®

1993 VISUAL BASIC 3.0 Microsoft®

1994 VISUAL BASIC 4.0 Microsoft®

1995 VISUAL BASIC 5.0 Microsoft®

1998 VISUAL BASIC 6.0 Microsoft®

1998 JAVA(JDK 1.2)

1998 JSP Sun Microsystems Uso en Internet (SERVIDOR).

+-1999 C# PROG. ORIENTADA A OBJETOS.

1999 Delphi 5

2000 JAVA(JDK 1.3) Sun Microsystems

2001 .NET Microsoft® PROG. ORIENTADA A OBJETOS.

2002 JAVA(JDK 1.4) Sun Microsystems

**12. Algunos lenguajes de programación**

**12.1. ADA**

Nombrado en honor de la primera persona programador de computadoras del

mundo, AUGUSTA ADA BYRON KING, Condesa de Lovelace, e hija del poeta inglés Lord

Byron.

Ada es un idioma de la programación de alto nivel pensado para las aplicaciones en vías

de desarrollo donde la exactitud, seguridad, fiabilidad y manutención son primeras metas.

Ada es un lenguaje del tipo orientado a objeto. Se piensa que trabaja bien en un ambiente

multi-lenguaje y ha estandarizado los rasgos para apoyar la unión a otros idiomas.

La Razón de Ada proporciona una descripción de los rasgos principales del idioma y

sus bibliotecas y explicaciones hacen lo propio con las opciones hechas por los diseñadores

del idioma.

**12.2. COBOL**

El deseo de desarrollar un lenguaje de programación que fuera aceptado por cualquier

marca de computadora reunió en Estados Unidos, en mayo de 1959, una comisión (denominada

CODASYL: Conference On Data Systems Languages) integrada por fabricantes de

computadoras, empresas privadas y representantes del Gobierno, dando lugar a la creación

del lenguaje COBOL (COmmon Business Oriented Language) orientado a los negocios,

llamándose esta primera versión COBOL-60, por ser este el año que vio la luz.

En sus inicios, Cobol estaba en constante evolución gracias a las aportaciones de usuarios

y expertos, dando lugar a revisiones en los años 1961, 1963 y 1965. La primera versión

estándar (ANSI) nació en 1968, siendo revisada en 1974 (COBOL ANS-74), en 1985 apareció

COBOL-ANS 85, en 1989 su versión ampliada (COBOL-ANSI) y en 2002 COBOL

ANS-2002. Cabe destacar que en 1997 Fujitsu lanzó una nueva versión, COBOL97. La última

versión comercializada de este lenguaje es Visual COBOL, del año 2011.

¿Por qué se hablaba de fabricantes de computadoras y no de sistemas operativos,

como en la actualidad? Sí que es significativo, pero por aquellos años no existían Sistemas

Operativos abiertos, sino que cada fabricante tenía el propio y por lo tanto cada Cobol

debería valer para cada computadora. Ciertamente no había mucha diferencia entre ellos.

Cobol es un lenguaje compilado, es decir, existe el código fuente escrito con cualquier

editor de textos y el código objeto (compilado) dispuesto para su ejecución con su correspondiente

runtime. Cuando se ve un programa escrito en Cobol saltan a la vista varios

aspectos:

— Existen unos márgenes establecidos que facilitan su comprensión.

— Está estructurado en varias partes, cada una de ellas con un objetivo

dentro del programa.

6-24

Desarrollo de Sistemas

— Nos recuerda mucho al idioma inglés, puesto que su gramática y su vocabulario

están tomados de dicho idioma.

— En contraste con otros lenguajes de programación, COBOL no se concibió para

cálculos complejos matemáticos o científicos; de hecho solo dispone de comandos

para realizar los cálculos mas elementales: suma, resta, multiplicación y división;

sino que su empleo es apropiado para el proceso de datos en aplicaciones

comerciales, utilización de grandes cantidades de datos y obtención de resultados,

ya sea por pantalla o impresos.

Con Cobol se pretendía un lenguaje universal, a pesar de lo cual los numerosos fabricantes

existentes en la actualidad han ido incorporando retoques y mejoras, aunque las

diferencias esenciales entre ellos es mínima.

Con la llegada del Sistema Operativo Windows, son muchos los que intentan proveer

al Cobol de esa interface gráfica: Objective Cobol, Visual Object Cobol de Microfocus, Fujitsu

PowerCobol, Acucobol-GT, Vangui y Cobol-WOW de Liant (RM), etc… que están consiguiendo

que este lenguaje siga estando presente en el modo visual de ofrecer los programas.

Sin embargo, son muchas las empresas que siguen dependiendo del Cobol-85

tradicional para sus proyectos debido principalmente a la estructura de su sistema informático.

**12.3. FORTRAN**

FORTRAN que originalmente significa Sistema de Traducción de Fórmulas Matemáticas

pero se ha abreviado a la FORmula TRANslation, es el más viejo de los establecidos

lenguajes de “alto-nivel”, fue diseñado por un grupo en IBM durante los años 50. El idioma

se hizo tan popular en los 60 que otros vendedores empezaron a producir sus propias

versiones y esto llevó a una divergencia creciente de dialectos (en 1963 había 40 recopiladores

diferentes).

Así las cosas, fue reconocido que tal divergencia no estaba en los intereses de los usuarios

o los vendedores, por lo que FORTRAN 66 fue el primer idioma en ser regularizado

oficialmente en 1972. La publicación de la norma significó que ese FORTRAN se llevó a

cabo más ampliamente que cualquier otro idioma. A mediados de los años setenta se proporcionó

virtualmente a cada computadora, mini o mainframe, un sistema FORTRAN 66

normal. Era, por tanto, posible escribir programas en FORTRAN en cualquier sistema y

estar bastante seguro de que estos pudieran moverse para trabajar en cualquier otro sistema

de forma bastante fácil. Esto hacía que pudieran procesarse programas de FORTRAN

muy eficazmente.

La definición normal de FORTRAN se puso al día en 1970 y una nueva norma, ANSI

X3.9-1978, fue publicada por el Instituto de las Normas Nacional americana. Esta norma

fue adoptada en 1980 por la Organización de Normas Internacionales (ISO)

como una Norma Internacional (ES 1539: 1980). El idioma es normalmente

conocido como FORTRAN 77 (desde que el proyecto final realmente se completó

en 1977) y es ahora la versión del idioma en su uso extendido.

6-25

Lenguajes de programación

El FORTRAN fue un lenguaje verdaderamente revolucionario, pues antes de él todos

los programas de computadores eran lentos y originaban muchos errores. En los primeros

tiempos, un programador podía escribir el algoritmo deseado como una serie de ecuaciones

algebraicas y el compilador FORTRAN podía convertir las declaraciones en lenguaje de

máquina que el computador podía reconocer y ejecutar.

El lenguaje FORTRAN original era muy pequeño en comparación con las versiones

modernas. Contenía apenas un número limitado de declaraciones tipo, y solo se podía trabajar

con el tipo “integer” (entero) y “real” (real), y tampoco había subrutinas. Cuando se

comenzó a usar este programa, se verificó la existencia de diversos errores, por lo que IBM

lanzó el FORTRAN II en 1958.

El desarrollo continuó en 1962, con el lanzamiento del FORTRAN IV. Este tenía muchas

mejoras y por eso se convirtió en la versión más utilizada en los quince años siguientes. En

1977 el lenguaje recibe otra actualización muy importante, incluyendo muchas nuevas características

que permitían escribir y guardar más fácilmente programas estructurados. El FORTRAN

77 introducía nuevas estructuras, como el bloque IF, y fue la primera versión del lenguaje

en que las variables “character” (caracteres) eran realmente fáciles de manipular. Este

lenguaje se volvió un poco limitado en términos de estructuras de información y también

por solo permitir la codificación de algunas figuras de programación estructurada.

La siguiente mejora fue más importante y dio origen al FORTRAN 90. Este incluía todo

el FORTRAN 77 como base pero con cambios significativos, fundamentalmente en las operaciones

sobre tablas (array) pero también sobre: una configuración en parámetros de las

funciones intrínsecas, permitiendo así utilizar una secuencia de caracteres muy grande,

como también usar más de dos tipos de precisión para variables del tipo Real y Complex;

se perfeccionó la computación numérica con la inclusión de un conjunto de funciones

numéricas, y mediante el desarrollo de un conjunto de funciones y subrutinas que permiten

acceder con mayor facilidad a bibliotecas de programas, función auxiliar en la definición

de datos globales; se mejoró la capacidad de escribir procedimientos internos y recursivos,

como también llamar los procedimientos a través de argumentos, siendo estos opcionales

u obligatorios; y se añadió una implementación del concepto de punteros.

En conjunto, los nuevos aspectos contenidos en FORTRAN 90 hacen que este sea

considerado como el lenguaje más eficiente de la nueva generación de supercomputadores

y aseguran que el FORTRAN continuará siendo usado con éxito por mucho tiempo.

FORTRAN 90 fue seguido de una pequeña mejora llamada FORTRAN 95, en 1997.

Este ofrece nuevas características del lenguaje, y clarifica algunas de las ambigüedades de

la antigua versión. Al FORTRAN 90 le siguió una pequeña mejora llamada FORTRAN 95

(en 1997). Posteriormente aparecieron FORTRAN 2003 y 2008, que introducen nuevas

características en el lenguaje.

**12.4. PASCAL**

El lenguaje de programación Pascal fue desarrollado originalmente por

Niklaus Wirth, un miembro de la International Federation of Information Processing

(IFIP). El Profesor Niklaus Wirth desarrolló Pascal para proporcionar

rasgos que estaban faltando en otros idiomas en aquel entonces.

6-26

Desarrollo de Sistemas

Los principales objetivos para Pascal eran: ser eficiente para llevarse a cabo y poder ejecutarse

los programas, permitir el desarrollo de estructuras y también organizar programas,

y servir como un vehículo para la enseñanza de los conceptos importantes de programación

de la computadora.

Pascal, que se nombró así gracias al matemático Blaise Pascal, es un descendiente

directo de ALGOL 60, qué ayudó a su desarrollo. Pascal también tomó componentes de

ALGOL 68 y ALGOL-W. El original idioma de Pascal aparecido en 1971 tuvo su última revisión

publicada en 1973.

En los años 80, la empresa de software Borland Software Corporation lanzó el compilador

Turbo Pascal para el IBM PC, orientado a su distribución masiva por su bajo precio

y porque permitía traducir código para diferentes arquitecturas de hardware. Con la versión

Turbo Pascal 5.5 se incorporó la programación orientada a objetos a Pascal, que

después se convirtió en el lenguaje de programación Delphi. Posteriormente a esta versión

5.5 aparecieron Turbo Pascal 6.0 en 1990 y Borland Pascal 7.0 en 1992.

**12.5. BASIC**

BASIC (Beginners All Purpose Symbolic Instruction Code), sistema desarrollado en la

Universidad de Dartmouth en 1964 bajo la dirección de J. Kemeny y T. Kurtz.

Se llevó a cabo para los G.E.225. Esto significa que es un idioma muy simple para

aprender y que es fácil de traducir. Además, los diseñadores desearon que fuera la base para

que los estudiantes aprendieran más adelante los idiomas más poderosos como FORTRAN

o ALGOL. Bill Gates y Paul Allen tenían algo diferente en mente.

En los 70 cuando la computadora personal Altair de M.I.T.S fue concebida, Allen convenció

a Gates a ayudarle a desarrollar un idioma básico para él. El futuro de BASIC y el

PC empezó. Gates estaba asistiendo a Harvard en ese momento y Allen era un empleado

de Honeywell.

Allen y Gates adoptaron su BASIC a M.I.T.S. para su Altair. Esta versión tomó un total

de 4k de memoria incluido el código y los datos que se usaron para el código fuente. Gates

y Allen pusieron a funcionar BASIC en otras plataformas.

En este momento la Corporación de Microsoft empezó su reinado en el mundo del

PC. Más tarde en los 70, BASIC se había puesto ya en plataformas como Apple, Comodor

y Atari y ahora era tiempo para el DOS de Bill Gates, que vino con un intérprete de BASIC.

La versión distribuida con MS-DOS era GW-BASIC y se ajustaba en cualquier máquina

que podía ejecutar DOS. No había ninguna diferencia entre BASIC-A y GW-BASIC, el

A proporcionado por las computadoras de IBM.

**12.6. C**

El lenguaje C reúne características de programación intermedia entre los

lenguajes ensambladores y los lenguajes de alto nivel; con gran poderío basa-

6-27

Lenguajes de programación

do en sus operaciones a nivel de bits (propias de ensambladores) y la mayoría de los elementos

de la programación estructurada de los lenguajes de alto nivel, por lo que resulta

ser el lenguaje preferido para el desarrollo de software de sistemas y aplicaciones profesionales

de la programación de computadoras.

En 1970, Ken Thompson de los laboratorios Bell se había propuesto desarrollar un

compilador para el lenguaje Fortran que corría en la primera versión del sistema operativo

UNIX tomando como referencia el lenguaje BCPL; el resultado fue el lenguaje B (orientado

a palabras) que resultó adecuado para la programación de software de sistemas.

Este lenguaje tuvo la desventaja de producir programas relativamente lentos. En 1971

Dennis Ritchie, con base en el lenguaje B, desarrolló NB que luego cambio su nombre por

C; en un principio sirvió para mejorar el sistema UNIX por lo que se le considera su lenguaje

nativo.

Su diseño incluyo una sintaxis simplificada, la aritmética de direcciones de memoria

(permite al programador manipular bits, bytes y direcciones de memoria) y el concepto de

puntero; además, al ser diseñado para mejorar el software de sistemas, se buscó que generase

códigos eficientes y una portabilidad total, es decir, que pudiese ejecutarse en cualquier

máquina. Logrados los objetivos anteriores, C se convirtió en el lenguaje preferido de

los programadores profesionales.

En 1980 Bjarne Stroustrup de los laboratorios Bell de Murray Hill, New Jersey, inspirado

en el lenguaje Simula67 adiciona las características de la programación orientada a

objetos (incluyendo la ventaja de una biblioteca de funciones orientada a objetos) y lo

denomina C con clases.

Para 1983 dicha denominación cambió a la de C++. Con este nuevo enfoque surge la

nueva metodología que aumenta las posibilidades de la programación bajo nuevos conceptos.

**13. Otros lenguajes de programación**

Cualquier notación para describir algoritmos y estructuras de datos se puede calificar

como un lenguaje de programación, pero principalmente este término se refiere a los

implementados para computadoras. Se han diseñado he implementado cientos de lenguajes

de programación distintos y actualmente todos ellos además cuentan con entornos

gráficos. Existen otros conceptos tomados en cuenta para agrupar los lenguajes, que dan

origen a diversas clasificaciones, entre los que destacan las siguientes:

— Lenguajes de cuarta generación 4GL: estos lenguajes se distinguen por formar

parte de un entorno de desarrollo, que comprende el controlador de una base

de datos y todo lo que de esto se deriva, como la administración de un diccionario

de datos, el control de accesos, el manejo de la consistencia de la información

y otras características enfocadas a facilitar los programas de acceso

y explotación de la información. Como ejemplos podemos citar a

los 4 grandes: PROGRESS, SYSBASE, INFORMIX, y ORACLE.

6-28

Desarrollo de Sistemas

— Lenguajes visuales: se les llama de esta manera a los lenguajes que forman parte

de una aplicación dotada de una interfaz gráfica, la cual por medio de iconos y

otras herramientas visuales y simbólicas, pretende facilitar las tareas rutinarias de

los programadores, como son el diseño y desarrollo de formularios e informes.

Los ejemplos más comerciales de estos lenguajes son: VISUAL BASIC, VISUAL

CAFE, VISUAL FOX, etc.

— Metalenguajes: son lenguajes como XML, SGML y HTML que sirven para definir

otros lenguajes, cuyo objetivo es llevar a cabo la estructuración de textos

mediante un conjunto de etiquetas, de manera tal, que puedan ser entendidos

por los humanos y también procesados por los ordenadores. Estos lenguajes

están teniendo un gran auge sobre la plataforma de Internet, en la cual son usados

para la creación de documentos y el intercambio o transferencia de información.

— Lenguajes de propósito específico: son aquellos lenguajes desarrollados con la

finalidad de resolver problemas de una naturaleza muy determinada, tal como

SPSS para problemas estadísticos, MATLAB para cálculos científicos y de ingeniería,

CAD/CAM para el diseño de piezas y programación de máquinas de control

numérico, como tornos y fresadoras, GPSS para simulación de sistemas,

CORBA para el manejo de interfaces en ambientes cliente-servidor, etc.

— Lenguajes script: son lenguajes como JAVASCRIPT, VBSCRIPT, PERLSCRIPT, que

se utilizan en ambientes clientes servidor, mediante la incrustación de código en

las páginas HTML, y así permitir la programación del lado del cliente, buscando,

fundamentalmente, hacer más atractivos las interfaces gráficas de las páginas.

— Lenguajes vinculados a Internet.

**14. Características, elementos y funciones de JAVA, C y C++**

A pesar de la evolución de los lenguajes de programación y el éxito de cada uno ellos,

hay mucho que avanzar todavía. Pero para comprender mejor esto último, analicemos las

características más esenciales de un buen lenguaje de programación:

1. Una forma clara, sencilla y exacta de la sintaxis para que el programador pueda

expresar las ideas de sus algoritmos (integridad conceptual), haciendo mas fácil

la comprensión del mismo para posibles mejoras o modificaciones.

2. Tener menos restricciones en la forma de codificación de los valores y en la colocación

de los mismos (ortogonalidad).

3. Tener la facilidad de que al codificar nuestro algoritmo podamos ver la parte

esencial del mismo en nuestro programa, es decir, al tener codificado nuestro

programa, podamos ver en él nuestro algoritmo una forma sencilla

para poder hacer modificaciones futuras (naturalidad de la aplicación),

pero también viendo como va quedando la estructura de

nuestros datos.

6-29

Lenguajes de programación

4. Apoyo para la abstracción, es decir, permitir al usuario una fácil creación de sus

estructuras de sus datos de forma breve y sencilla, sin caer en redundancias.

5. Un gran problema para los lenguajes de programación es la facilidad de verificar

los programas, pero sobre todo la confiabilidad de que cuando se hace un programa,

el lenguaje verifique todos los posibles errores de semántica y a su vez que

sus datos de salida, o casos de entrada, sean reales y confiables.

6. Otro punto es el entorno de programación. Sin una interfaz, una ayuda o herramientas

para la programación, el lenguaje sería inútil, aburrido y desesperante

para el programador.

7. Un factor importante en la creación de los programas es la portabilidad de los

mismos hacia otros sistemas. Para ello el lenguaje de programación debe permitirlo

por medio de no basarse en una sola arquitectura en la ejecución de los programas,

tal es el caso de C, FORTRAN, Ada y Pascal, que manejan la implementación

de los programas hacía otros sistemas.

8. El coste según su uso. El coste de tiempo en la ejecución, de traducción, de creación,

prueba y uso y coste de mantenimiento de los programas.

Otras características de los lenguajes de programación son los entornos de diseño.

Estos pueden ser:

— Entorno de procesamiento por lotes. Las instrucciones se ejecutan por secciones,

estructuras o lotes.

— Entorno interactivo. Posibilidad de uso de los periféricos.

— Entornos incrustados. Son aquellos lenguajes grandes que tienen la posibilidad

de llamar a otros más pequeños.

— Entornos de programación.

— Entornos de marcos de ambiente. Posibilidad de interactuar con la red.

Un punto importante en todas estas características es que se debe tener un estándar

para evitar caer en incompatibilidades de equipo.

**14.1. Introducción histórica: C, C++ y JAVA**

En la década de los setenta Ken Thompson creó un lenguaje de programación denominado

B que no tubo repercusiones espectaculares pero que sirvió de base para que Dennos

Ritchie en esta misma década crease el lenguaje C que a finales de los

setenta y durante los ochenta el fue el lenguaje de programación más utilizado

por los programadores. Este lenguaje se desarrolló usando UNIX como sistema

operativo. En 1983 se estableció un comité para crear el estándar ANSI

6-30

Desarrollo de Sistemas

que definiera el lenguaje C. Al cabo de seis años, 1989, este estándar fue adoptado comenzando

su implementación en 1990. En este mismo año este estándar también fue adoptado

por la ISO.

En 1995 se adoptó la Enmienda 1 del estándar C. En 1989 se creó un nuevo estándar

que, junto con la Enmienda 1, se convirtió en el documento base del estándar C++. A

partir de este momento C quedó relegado a un segundo plano, pero totalmente operativo.

De hecho, en 1999, se creó un nuevo estándar y en la actualidad sigue siendo un lenguaje

de programación en pleno vigor (a este estándar se le suele denominar C99).

Es, por tanto, la creciente complejidad de los sistemas lo que ha conducido a la necesidad

de cambiar de C a C++. C++ fue inventado por Bjarne Stroustrup en 1979 en los

Laboratorios Bell. La nueva forma de pensar (programación orientada a objetos) hizo que

fuese una auténtica revolución en el mundo de los lenguajes de programación. C++ es una

extensión de C en la que se añaden las características orientadas a objetos, fue estandarizado

en 1997 cumpliendo los estándares ANSI/ISO.

C++ fue el lenguaje dominante a finales de los ochenta y principios de los noventa.

No obstante, y aunque combinaba de forma perfecta la programación orientada a objetos

con un lenguaje tan completo como C, llegó Internet y se hizo necesario adaptar este lenguaje

a los nuevos. Así fue como en 1995 vio la luz JAVA, aunque ya en 1991 fue desarrollado

y comenzó a ser utilizado. Fue desarrollado durante año y medio por cinco programadores

expertos que crearon este lenguaje de programación cuya sintaxis básica esta

basada en C y que implementa la orientación a objetos de C++. El dominio de JAVA como

lenguaje de programación orientado a objetos se ha extendido hasta ahora. Su utilidad

tanto para aplicaciones independientes de la plataforma como para subprogramas ejecutables

dentro de una pagina web (applets), y muchas características más que veremos a continuación,

le han hecho digno de todo elogio.

**14.2. El lenguaje C**

El lenguaje C es el resultado de un proceso de desarrollo que se inició con un lenguaje

denominado BCPL. Este influenció a otro llamado B (inventado por Ken Thompson). En

los años 70, este lenguaje llevó a la aparición del C.

Con la popularidad de las microcomputadoras muchas compañías comenzaron a

implementar su propio C por lo cual surgieron discrepancias entre sí.

Por esta razón ANSI estableció un comité en 1983 para crear una definición no ambigüa

del lenguaje C e independiente de la máquina que pudiera utilizarse en todos los tipos

de C. Algunos de las C existentes son:

— Quick C.

— C++.

— Turbo C.

— Turbo C ++.

6-31

Lenguajes de programación

— Borland C.

— Borland C++.

— Microsoft C.

Cuando se habla del lenguaje C se ha de tener en cuenta las dos grandes estandarizaciones

existentes en su larga historia: por un lado tenemos C89 y C99. Para hacernos una

idea de las diferencias, C89 contiene 32 palabras clave, C99 incluye cinco más. Hoy en día

la mayoría de los compiladores se basan en esta segunda estandarización.

El lenguaje C se engloba dentro de los lenguajes de nivel medio. Como lenguaje de

nivel medio, C permite la manipulación de bits, bytes y direcciones que son los elementos

básicos con los cuales funciona la computadora. El lenguaje C es muy portable, en el sentido

de que funciona en distintos sistemas o diferentes tipos de computadoras; para que

nos hagamos una idea: Windows en todas sus versiones, DOS, Linux, etc.

C no lleva a cabo una comprobación de errores en tiempo de ejecución. Es el programador

el único responsable de llevar a cabo estas comprobaciones.

Dado que con C podemos manipular bits, bytes y direcciones, se hace ideal para la

programación de sistemas. De hecho el sistema Linux nació con el intento logrado de reescribir

el código de UNIX en C.

C es un lenguaje estructurado, pero no estructurado en bloques, ya que no permite

por ejemplo la creación de funciones dentro de funciones. Al ser, por tanto, un lenguaje

estructurado, su sintaxis es la que sirve de base para C++ y JAVA y será la que veamos en

un apartado posterior en este mismo tema.

El componente principal de C es la función que se define como una subrutina independiente.

Cada una de estas funciones está formada por bloques en los que se desarrolla

toda la actividad del programa. Cada bloque queda delimitado por “{“ y “}”. A estos bloques

se les conoce como bloques de código que son un grupo de instrucciones de un programa

conectados de forma lógica y que es tratado como una unidad.

El lenguaje C tiene una lista de palabras clave que tienen un objetivo definido. Cada

una de estas palabras clave no puede ser utilizada para otro fin diferente al que tiene asignado.

En C, a pesar de no ser un lenguaje fuertemente tipado como ocurre con C++ y JAVA,

sí que se diferencia entre mayúsculas y minúsculas.

Como hemos visto, en C un programa consiste en una o más funciones; no obstante

siempre ha de haber una que sea invocada cuando se ejecuta el programa, es decir, que sea

la primera que se ejecute y que llame a las demás (una función principal), a esta función se

le ha de denominar main(). Main no es una palabra reservada pero no puede

usarse para otras cosas (trataremos main como si fuese una palabra reservada

aunque no lo sea).

6-32

Desarrollo de Sistemas

La mayoría de los programas C incluyen llamadas a varias funciones contenidas en la

biblioteca estándar de C. Todos los compiladores actuales de C incorporan esta biblioteca

estándar en la cual se incluyen funciones que realizan las tareas más habituales (raíces,

impresiones, lectura de ficheros, etc.). Independientemente de esta biblioteca estándar

existen muchas otras que podemos utilizar, o incluso crear nuestra propia biblioteca de

funciones. Para poder fusionar el código del programa con el código de las funciones existentes

en las bibliotecas los compiladores incluyen enlazadores, a cuyo proceso se le denomina

“enlace”.

En C un programa se puede escribir en varios archivos y compilar cada uno de ellos

por separado. De este modo la recompilacion se puede efectuar en aquel archivo que da el problema o en el cual queramos realizar alguna modificación sin tener que recompilar todo el programa. El código objeto completo lo forman todos los archivos del programa compilados y las rutinas de las bibliotecas utilizadas.

Por tanto los pasos a seguir en la creación de un programa C son:

***— Creación del programa.***

***— Compilación del programa.***

***— Enlace del programa con todas las funciones que se necesiten de la biblioteca.***

**14.2.1. Elementos generales de un programa en C**

Aunque cada uno de los programas son distintos, todos tienen características comunes.

Los elementos de un programa en C son los siguientes:

Comentarios

Inclusión de archivos

main()

{

variables locales

flujo de sentencias

}

Definición de funciones creadas por el programador utilizadas en main()

Veamos en qué consiste cada uno:

— Comentarios: se identifican porque van entre diagonales y asterisco.

Nos sirve para escribir información que nos referencie al programa

pero que no forme parte de él. Por ejemplo, especificar qué hace el

programa, quién lo elaboró, en qué fecha, qué versión es, etc.

6-33

Lenguajes de programación

— Inclusión de archivos: consiste en llamar a la o las bibliotecas donde se encuentran

definidas las funciones de C (instrucciones) que estamos utilizando en el

programa. En realidad, la inclusión de archivos no forma parte de la estructura

propia de un programa sino que pertenece al desarrollo integrado de C. Se incluye

aquí para que el alumno no olvide que debe llamar a los archivos donde se

encuentran definidas las funciones estándar que va a utilizar.

— main(): en C todo está constituido a base de funciones. El programa principal no

es la excepción. main() indica el comienzo de la función principal del programa,

la cual se delimita con llaves.

— Variables locales: antes de realizar alguna operación en el programa, se deben declarar la(s) variable(s) que se utilizarán en el programa.

— Flujo de sentencias: es la declaración de todas las instrucciones que conforman

el programa.

— Definición de funciones creadas por el programador utilizadas en main(): finalmente,

se procede a definir el contenido de las funciones utilizadas dentro de

main(). Estas contienen los mismos elementos que la función principal.

Un programa en C consta de tres secciones. La primera sección es donde van todos

los headers. Estos headers son comúnmente los #define y los #include. Como segunda sección

se tienen las funciones. Al igual que Pascal, en C todas las funciones que se van a ocupar

en el programa deben ir antes que la función principal (main()). Declarando las funciones

a ocupar al principio del programa, se logra que la función principal esté antes que

el resto de las funciones. Ahora, solo se habla de funciones ya que en C no existen los procedimientos.

Y como última sección se tiene a la función principal, llamada main. Cuando

se ejecuta el programa, lo primero que se ejecuta es esta función, y de ahí sigue el resto

del programa.

Los símbolos { y } indican begin y end respectivamente. Si el contenido de una función

o de un ciclo while, por ejemplo, es de solamente una línea, no es necesario usar llaves

({ }), en caso contrario es obligatorio usarlas.

Ejemplo de un programa en C

/\*Programa que imprime un saludo en pantalla\*/

#include <stdio.h>

tomates () {

printf("Dedicado a Ismael”);

}

void main() {

tomates();

}

/\* Fin programa \*/

6-34

Desarrollo de Sistemas

Los primeros lenguajes ensambladores ofrecen una forma de trabajar directamente

con un conjunto de instrucciones incorporadas en la computadora. Cada una de estas

especificaciones se ha de especificar en términos de máquina (bits de los registros).

Dado que esto se hacía muy pesado para el programador, surgieron los primeros lenguajes

de alto nivel, como Fortran, que en un principio se desarrollaron como alternativa

a los lenguajes ensambladores. En un principio estos lenguajes fueron usados para resolver

problemas de matemáticas, ingeniería o científicos (lenguajes orientados al problema).

El lenguaje C está ligado a la computadora y nos ofrece un importante control sobre

los detalles de la implementación de una aplicación; esta es la razón por la cual se le considera

a la vez un lenguaje de bajo y de alto nivel (lenguaje de nivel medio).

Entre las muchas ventajas que posee el lenguaje C citaremos:

— Tamaño óptimo de código. Dado que tiene pocas reglas de sintaxis.

— Conjunto de palabras clave. Palabras reservadas que usa el lenguaje.

— Ejecutables rápidos. Muchos programas C se ejecutan con una velocidad equivalente

al lenguaje ensamblador.

— Comprobación de tipos limitada. Se permite visualizar datos de distintas maneras.

— Implementación de diseño descendente. El denominado diseño Top-Down gracias

a la implementación de sentencias de control.

— Estructura modular. Compilación y enlazado por separado.

— Interfaz transparente para el lenguaje ensamblador. Se puede llamar a las rutinas

del lenguaje ensamblador desde un compilador C.

— Manipulación de bits. C permite manipular bits y bytes.

— Tipos de datos puntero. C permite manipular direcciones.

— Estructuras extensibles. Los arrays son unidimensionales.

— Memoria eficiente. Los programas C tienden a ser muy eficientes en memoria.

— Portabilidad entre plataformas. Un programa C se puede ejecutar en una computadora

u otra con un sistema operativo u otro.

— Rutinas de biblioteca. Hay una gran cantidad de bibliotecas con funciones precreadas.

(Para más información sobre el Lenguaje C, consulte el Anexo I de este

mismo tema).

6-35

Lenguajes de programación

**14.3. El lenguaje C++**

C++ es un subconjunto de C que mantiene todas las características de este y su flexibilidad

para el trabajo en el tratamiento de la interfaz hardware/software, su programación

del sistema a bajo nivel, sus expresiones, etc., pero todo ello dentro de la programación

orientada a objetos.

Este lenguaje combina las construcciones del lenguaje procedimental estándar y el

modelo orientado a objetos. Se trata pues de una nueva forma de pensar.

C++ se desarrolló originariamente para resolver simulaciones conducidas por sucesos.

Fue utilizado en 1983 y aún en 1987 se encontraba en fase de evolución. En su evolución

siempre se ha procurado preservar la integridad de los programas escritos en otros lenguajes

al intentar exportarlos a C++.

Diferencias entre C y C++:

— Trabajo con clases. Frente al trabajo con estructuras definido en C.

— Constructores de clases y encapsulación de datos.

— La clase struct. Esta clase puede contener tanto datos como funciones.

— Constructores y destructores. Se usan para garantizar la inicialización y destrucción

de los datos.

— Mensajes. Los objetos se manipulan enviándoles mensajes.

— Funciones afines. Se permite acceder a los métodos y datos de una clase privada.

— Sobrecarga de operadores. Se puede hacer sobrecarga de operadores por número

de argumentos o por su tipo.

— Clases derivadas. Una subclase de una clase específica.

— Polimorfismo. El objeto determina qué clase o subclase recibe un mensaje.

— Biblioteca de flujos. Permitiendo que las operaciones de entrada y salida de datos

desde un terminal o un archivo sean más accesibles.

Existen otras diferencias menos notorias como:

— Su sintaxis en algunos aspectos puntuales como son los comentarios y variables

enumeradas

— Conversiones de tipo explícitas.

— Sobrecarga de funciones.

— Argumentos por referencia.

— Punteros de tipo void.

— Funciones inline, etc.

6-36

Desarrollo de Sistemas

Ejemplo de programa en C++:

/\* Comentario creado para los estudiantes de TAI, mucha suerte a todos \*/

# include <stdio.h>

int main()

{

printf(“C++ es guay”);

return (0);

}

Este código fuente ha de ser guardado con extensión \*.c, luego será compilado para

poder ejecutarlo.

Existen entornos gráficos para el desarrollo de programas en C y en C++ que incluyen

los compiladores correspondientes y facilitan visualmente la labor del programador.

En la imagen vemos el Visual C++ de Microsoft que permite escribir programas C o

C++.

(Para más información sobre el Lenguaje C++, consulte el Anexo II de

este mismo tema).

6-37

Lenguajes de programación

**14.4. El lenguaje JAVA**

JAVA es un lenguaje de programación desarrollado por un grupo de ingenieros de Sun

Microsystems (1991); en principio está destinado a electrodomésticos, está basado en C++

y se diseñó para ser un lenguaje sencillo con códigos de tamaño muy reducido. Posteriormente

(1995) se comienza a utilizar como lenguaje para computadores; Netscape Navigator

incorpora un intérprete JAVA en su versión 2.0 siendo la base para JAVAScript. El rápido

crecimiento de Internet y el uso de JAVA para dar dinamismo a las páginas de HTML,

lo convierten en un medio popular de crear aplicaciones para Internet. Si bien su uso se

destaca en el web y sirve para crear todo tipo de aplicaciones (locales, Intranet o Internet).

En la actualidad es un lenguaje muy completo (la versión JAVA 1.0 tenía 12 paquetes

(packages), JAVA 1.1 tenía 23 y JAVA 1.2 o JAVA 2 tiene 59). El haber sido diseñado en una

época muy reciente y por un único equipo le confieren unas características que facilitan su

aprendizaje y utilización a los usuarios; JAVA incorpora muchos aspectos que en cualquier

otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores

(threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.).

La importancia de JAVA es su utilización como nexo de unión de los usuarios con la

información, ya sea en el ordenador local, en un servidor de web, en una base de datos o

en cualquier otro lugar.

JAVA es un lenguaje potente que resuelve los problemas que se plantean al acceder a

una base de datos, en la programación de redes, la distribuida, etc.; tiene muchas posibilidades

de utilización como aplicación independiente (Standalone Application), programación

a través de los applets, ejecución como servlet, etc. Un applet es un programa que

corre bajo un navegador o browser (por ejemplo Netscape Navigator o Internet Explorer)

y es descargado como parte de una página HTML desde un servidor web. El applet se descarga

desde el servidor y no requiere instalación en el ordenador donde se encuentra el

browser. Un servlet es una aplicación sin interface gráfica que se ejecuta en un servidor de

Internet.

Es un lenguaje orientado a objetos, ha sido concebido como tal a diferencia de otros

lenguajes como C++ que son lenguajes modificados para poder trabajar con objetos.

**14.4.1. ¿Qué entendemos por objeto?**

Podemos decir que todo puede verse como un objeto. Un objeto es una pieza de software

que cumple con ciertas características:

— Encapsulamiento: el objeto es autocontenido, es la integración de sus datos (atributos)

y los procedimientos (métodos) que actúan sobre él. Al utilizar la programación

orientada a objetos, se definen clases (objetos genéricos) y la forma en

que interactúan entre ellos, a través de mensajes. Dado que los programas

no modifican al objeto, este se mantiene independiente del

resto de la aplicación; si necesitamos modificar un objeto lo hacemos

sin tocar el resto de la aplicación.

6-38

Desarrollo de Sistemas

— Herencia: se pueden crear nuevas clases que comparten características (atributos)

y comportamientos (métodos) de otras ya preexistentes, relacionadas por

una relación jerárquica, simplificando la programación.

JAVA es independiente de la plataforma, puede hacerse funcionar con cualquier ordenador.

Al compilar un programa JAVA, lo que se genera es un pseudocódigo definido por Sun,

para una máquina genérica; el software de ejecución java interpreta las instrucciones, emulando

a dicha máquina. Por supuesto esto no es muy eficiente, por lo que tanto Netscape

como Hot JAVA o Explorer, al ejecutar el código por primera vez, lo van compilando (mediante

un JIT: Just In Time compiler), de modo que al crear por ejemplo la segunda instancia de

un objeto, el código ya está compilado específicamente para la máquina huésped.

**14.4.2. Compilador de Java, JAVA Virtual Machine**

Existen distintos programas comerciales que permiten desarrollar código JAVA. Sun

distribuye gratuitamente el JDK (JAVADevelopment Kit); el JDK es un conjunto de programas

y librerías que permiten desarrollar, compilar y ejecutar programas en JAVA.

El Compilador de JAVA realiza un análisis de sintaxis del código escrito en los ficheros

fuente de JAVA (con extensión \*.java). Si no encuentra errores en el código genera los

ficheros compilados (con extensión \*.class). En otro caso muestra la línea o líneas erróneas.

Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la

ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las

variables.

Los IDEs (Integrated Development Environment-Entornos de Desarrollo Integrados),

permiten en un mismo programa escribir el código JAVA, compilarlo y ejecutarlo sin tener

que cambiar de aplicación. Estos entornos integrados permiten desarrollar las aplicaciones

de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya

desarrollados, los cuales se incorporan al proyecto o programa. Como inconvenientes se

pueden señalar algunos fallos de compatibilidad entre plataformas y ficheros resultantes de

mayor tamaño que los basados en clases estándar.

La existencia de distintos tipos de procesadores y ordenadores resalta la importancia

de contar con un software que no dependa del tipo de procesador utilizado. Esto llevó a

los ingenieros de Sun a desarrollar un código capaz de ejecutarse en cualquier tipo de

máquina. Al ser compilado el código fuente no necesita ninguna modificación al cambiar

de procesador o al ejecutarlo en otra máquina; esto se debe a que se ha desarrollado un

código “neutro” el cual estuviera preparado para ser ejecutado sobre una “máquina hipotética

o virtual”, denominada JAVA Virtual Machine (JVM)). La JVM interpreta el código

neutro y lo convierte en el código particular de la CPU utilizada, evitando tener que realizar

un programa diferente para cada CPU o plataforma.

Nota. JDK, por tanto, es necesario si queremos crear applets o aplicaciones en

lenguaje JAVA y JVM es necesario si queremos visualizarlos. JDK, no obstante,

es un kit de desarrollo que lleva integrado JVM.

6-39

Lenguajes de programación

**14.4.3. Estructura de un programa JAVA**

La estructura de un programa realizado en cualquier lenguaje orientado a objetos (Object Oriented Programming) (OOP-POO), y en particular en el lenguaje JAVA, es una clase. En JAVA todo forma parte de una clase, es una clase o describe cómo funciona una clase. El conocimiento de estas es fundamental para poder entender los programas Java. Todas las acciones de los programas JAVA se colocan dentro del bloque de una clase o un objeto. Todos los métodos se definen dentro del bloque de la clase, ***JAVA no soporta funciones o variables globales.***

***En todo programa nos encontramos con una clase que contiene el programa principal y algunas clases de usuario*** (las específicas de la aplicación que se está desarrollando) que son utilizadas por el programa principal.

Los ficheros fuente tienen la extensión \*.java, mientras que los ficheros compilados tienen la extensión \*.class.

Un fichero fuente (\*.java) puede contener más de una clase, pero solo una puede ser publico.

El nombre del fichero fuente debe coincidir con el de la clase public (con la extensión \*.java). Si, por ejemplo, en un fichero aparece la declaración (public class MiClase {...}) entonces el nombre del fichero deberá ser MiClase.java. Es importante que coincidan mayúsculas y minúsculas ya que MiClase.JAVA y miclase.JAVA serían clases diferentes para Java.

Si la clase no es public, no es necesario que su nombre coincida con el del fichero.

Una clase puede ser public o package (default), pero no private o protected.

***En general una aplicación está constituida por varios ficheros \*.class.***

Cada clase realiza unas funciones particulares, permitiendo construir las aplicaciones con gran modularidad e independencia entre clases.

Las clases de Java se agrupan en **packages**, que son **librerías de clases.** Si las clases no se definen como pertenecientes a un package, se utiliza un package por defecto (default) que es el directorio activo.

Es necesario entender y dominar la sintaxis utilizada en la programación; observemos nuestro primer programa en Java y un breve comentario de las partes que lo componen, para posteriormente pasar a estudiar la nomenclatura empleada y los elementos que empleamos para desarrollar nuestro lenguaje:

/\* lolo.JAVA

Escribe en pantalla "¡Dedicado a Maria Luz!" \*/

class lolo{

public static void main(String args [ ]){

System.out.println(" ¡Dedicado a Maria Luz!") ;

}

}

Con JAVA se pueden crear dos tipos de programas: aplicaciones y applets. Una aplicación es un programa que se ejecuta en una computadora utilizando el sistema operativo de esa computadora. Se trata pues de un programa normal como podría haber sido en C o C++ pero en el lenguaje JAVA. En este aspecto la funcionalidad de JAVA no es diferente a la de cualquier otro lenguaje orientado a objetos. Una applet es una aplicación diseñada para ser transmitida por Internet y ejecutada en un navegador web compatible con JAVA.

Un applet es realmente un pequeño programa que se transfiere dinámicamente a través

de la red, como si fuese una imagen, un archivo de sonido o de vídeo. La diferencia principal

es que una applet es un programa que puede reaccionar ante las acciones del usuario

y cambiar dinámicamente.

Vamos a detallar algunos de los aspectos (cualidades) que han hecho de JAVA uno de

los lenguajes más populares:

**• Simple**

JAVA es un lenguaje relativamente fácil de aprender una vez que se comprenden los

conceptos básicos de la programación orientada a objetos. Además, para la realización de

una determinada acción existen siempre varios caminos por los cuales podamos programar.

**• Seguro**

Esta cualidad hace que los applets sean ideales para la transmisión por Internet sin violar

la vulnerabilidad de los sistemas.

**• Portable**

El código ejecutable generado por JAVA es adaptable a cualquier tipo de plataforma

con cualquier tipo de sistema operativo.

**• Orientado a objetos**

JAVA junto con C++ es el máximo exponente de la programación orientada a objetos.

Esta cualidad le brinda de todas las ventajas de esta programación.

**• Robusto**

Ya que JAVA se ejecuta en multitud de plataformas, esta cualidad le permite satisfacer

con éxito su deber en todas ellas.

**• Multihilo**

JAVA permite que se ejecuten varios hilos al mismo tiempo (cada hilo

representa una tarea a desarrollar). Esta cualidad es la que le ha hecho ideal

para trabajo en redes. Además permite la sincronización de cada una de esas

tareas y su comunicación.

6-41

Lenguajes de programación

**• Arquitectura neutral**

Esta cualidad le permite que un programa JAVA se pueda ejecutar en un sistema operativo

actual y en las actualizaciones que puedan surgir de ese sistema operativo. Lo único

que necesitamos es la JVM (máquina virtual de JAVA).

**• Interpretado**

Una vez compilado un programa este se convierte en código binario que es interpretado

por un intérprete de Java. Este intérprete debe ser optimizado para que la conversión

al código máquina sea de buen rendimiento.

**• Distribuido**

A través del protocolo TCP/IP podemos distribuir programas JAVA en Internet. JAVA

dispone además de RMI (Invocación de Método Remoto) que permite ejecutar procedimientos

de forma remota. Otra característica es la programación cliente/servidor.

Existen entornos gráficos para el desarrollo de programas en JAVA que incluyen los

compiladores correspondientes y facilitan visualmente la labor del programador.

(Para más información sobre el Lenguaje JAVA, consulte el Anexo III de este mismo

tema).

**15. Entornos de programación visual**

El desarrollo de interfaz de tipo gráfico en los sistemas operativos impulsó el desarrollo

de aplicaciones que utilizaran dicho entorno. A partir de 1990, con la proliferación de

Windows, esto se hizo especialmente importante.

Visual Basic surgió a principios de los años 90. Basado en el popular lenguaje Basic

supuso una auténtica revolución en el mundo de la programación, entre otras cosas por su

facilidad de uso y su entorno eminentemente gráfico. Las primeras versiones del lenguaje

no eran excesivamente potentes, pero las últimas (5.0 y 6.0) incorporan todo tipo de

herramientas y controles que permiten la construcción de aplicaciones de desarrollo en

cualquier entorno.

Todas estas herramientas ofrecen una gran facilidad en la construcción de aplicaciones.

Están orientadas tanto a eventos como a objetos, y proporcionan la posibilidad de

desarrollar aplicaciones para Windows sin necesidad de utilizar primitivos lenguajes para

programar directamente la interfaz gráfica.

Por el contrario, el código generado no está completamente optimizado

y suele tener un tamaño superior al de otros programas, por lo que su velocidad

de ejecución es más lenta que en entornos no gráficos.

6-42

Desarrollo de Sistemas

**15.1. Visual Basic**

Cuando se inicia Visual Basic aparece una ventana denominada IDE (Entorno Integrado

de Desarrollo) a través de la cual desarrollaremos la aplicación.

La ventana del IDE contiene las restantes ventanas del entorno de desarrollo. Dicha

ventana usa un entorno MDI, es decir, permite mantener múltiples ventanas abiertas contenidas

en una ventana principal. En un proyecto Visual Basic tenemos tres formas principales

de trabajar, diseño, ejecución e interrupción:

— En la fase de diseño se incorporan todos los controles, seleccionados a través del

cuadro de herramientas, que van a formar parte de la aplicación. El objeto que

va a contener todos los controles se denomina formulario. Una vez que se han

incorporado los controles se pueden establecer sus propiedades de dos formas:

mediante la ventana de propiedades del control o mediante código.

— En la fase de ejecución se escriben las sentencias de código necesarias, utilizando

la sintaxis del lenguaje, para realizar acciones, modificar propiedades de un

objeto o invocar a sus métodos.

— En la fase de interrupción se depuran los posibles errores que contenga el código.

En un proyecto puede haber más de un formulario. La combinación de formularios

módulos de código, clases y otros recursos son parte integrante de un proyecto de Visual

Basic. Normalmente los proyectos Visual Basic tienen extensión EXE.

Clarifiquemos toda la exposición anterior observando el IDE de Visual Basic:

6-43

Lenguajes de programación

Visual Basic es un lenguaje estructurado aunque permite utilizar estructuras de tipo

goto en ciertas rutinas, como las de tratamiento de errores.

Otras características importantes del lenguaje VISUAL BASIC son:

— Contiene una biblioteca de clases que da soporte a los objetos Windows tales

como:

• Ventanas.

• Cuadros de texto.

• Botones de pulsación.

• Casillas de verificación.

• Listas desplegables.

• Listas combinadas.

• Marcos.

• Etiquetas.

— Tiene un entorno de desarrollo integrado que incluye, entre otras:

• Editor de texto.

• Intérprete.

• Depurador.

• Examinador de objetos.

• Explorador de proyectos.

• Compilador.

— Dispone de asistentes para:

• Aplicaciones.

• Barras de herramientas.

• Formularios de datos.

• Empaquetado y distribución.

• Crear la interfaz pública de controles ACTIVEX.

• Objetos de datos.

• Generador de clases.

• Diseñador de complementos.

6-44

Desarrollo de Sistemas

— Galería de objetos vinculados e incrustados.

— Creación de bibliotecas dinámicas.

— Soporte para:

• Aplicaciones de Internet.

• Estandar.com.

— Acceso a base de datos utilizando:

• Controladores ODBC.

• Motor de Access.

• OLEDB.

• Controles ADO y DATA.

— Biblioteca para SQL que permite la manipulación de base de datos relacionales.

— Unadministrador visual de datos para manipular base de datos.

— Utilidad para crear ficheros de ayuda estilo Windows.

Mediante la implementación de todas estas características mencionadas es posible

realizar cualquier tipo de desarrollo utilizando Visual Basic. Con herramientas como el

Dataenvironment todo el proceso de acceso a base de datos es de muy sencilla utilización,

ya que, al crear una nueva conexión, aparece automáticamente un asistente para poder

seleccionar el proveedor de acceso a los datos, su localización y otros parámetros dependientes

del tipo de controlador. También, como se ha mencionado, es posible utilizar

herramientas que generan código SQL.

Como contrapunto a todas estas funciones, Visual Basic no ofrece demasiadas herramientas

relacionadas con Internet. Dispone solamente de dos controles: WINSOCK e

INET. Con el primero se pueden desarrollar servicios de comunicaciones tanto servidor

como cliente, tomando como base los servicios. El segundo control facilita la comunicación

entre servidores de tipo FTP y http, lo que es adecuado para crear clientes que necesiten

transferencia de archivos y descargas de documentos en la web.

**15.2. Otros entornos visuales**

Existen otros entornos de programación visual aparte de los ya mencionados. Otras

herramientas RAD (Herramientas de Desarrollo Rápido) son:

— Borland C++ Builder.

— Power++.

— Sybase Powerbuilder.

6-45

Lenguajes de programación

Borland C++ Builder usa prácticamente el mismo entorno que Delphi. Al igual que

Delphi está abierta a otras tecnologías. La mayor ventaja que ofrece es que el lenguaje utilizado

para programar es C++, lenguaje muy potente y versátil.

El entorno POWER++, de Sybase, dispone de un compilador muy optimizado y de un

entorno de trabajo muy eficiente. También tiene capacidades para crear aplicaciones cliente/

servidor y aplicaciones web, aunque su uso no está demasiado extendido en nuestro

país.

POWERBUILDER, también de Sybse, está relacionado estrechamente con el mundo

de las bases de datos. Es un lenguaje de cuarta generación que, al igual que los demás, se

apoya en el uso de objetos. Su gran ventaja es su capacidad multiplataforma.

Pero sin duda, las herramientas que más desarrollo han tenido dentro del mundo RAD

han sido todas aquellas relacionadas con JAVA. Casi la totalidad de los entornos de desarrollo

de JAVA son RAD y disponen de avanzados conjuntos de componentes y generadores

de código. Mencionaremos algunas como: JBUILDER, IBM VISUALAGE FOR JAVA,

SUN JAVAESTUDIO y MICROSOFT VISUAL J++.

**16. .NET Framework**

**16.1. Introducción**

Tal como entendíamos los entornos de desarrollo hasta “ayer”, estos permitían desarrollar

aplicaciones para DOS, para Linux, para Windows, etc. Si queríamos desarrollar

una aplicación con C++, debíamos ejecutar Visual C++; si queríamos desarrollar una aplicación

ASP, podíamos ejecutar Microsoft InterDev; si queríamos desarrollar una aplicación

en Visual Basic, debíamos desarrollar en el entorno de desarrollo de Visual Basic, etc.

Sin embargo, .NET Framework ha modificado esta idea por completo, y otras empresas

de creación de software de desarrollo tienden hacia esta idea.

**16.2. ¿Un mismo entorno para todos los lenguajes?**

En realidad el cambio conceptual no es tan complicado de comprender. .NET Framework

comparte la misma estructura general para todos los lenguajes de desarrollo. Imaginemos

una mano, tiene cinco dedos, cada dedo se llama de manera diferente y tiene unas

características particulares, uno es más largo, el otro más gordo, otro el más pequeño... sin

embargo, todos comparten el mismo corazón, el mismo cerebro y el mismo brazo; es un

ejemplo un poco simple, pero tiene todo el sentido, como se verá en las siguientes explicaciones.

Así por ejemplo, y hablando de .NET Framework en concreto, dentro de un entorno

de desarrollo o trabajando dentro del marco de trabajo.NET, podemos trabajar

con diferentes lenguajes de desarrollo, es decir, podemos trabajar con

JScript .NET, Visual Basic .NET, Visual C#, ASP .NET, etc. Cada lenguaje posee

sus propias características que permiten al desarrollador trabajar con el que

6-46

Desarrollo de Sistemas

más conozca, el que más le guste o con el que se sienta más identificado o cómodo.

Utilizando un mismo entorno de desarrollo como Visual Studio .NET, podemos

emplear en nuestros desarrollos el lenguaje o lenguajes que consideremos oportuno. En

realidad, conviene antes de seguir, comprender la diferencia entre Visual Studio .NET y

.NET Framework, ya que muchos desarrolladores confunden las diferencias existentes

entre estos dos.

.NET Framework es el marco de trabajo con el cual desarrollaremos nuestras aplicaciones.

En él se incluyen las diferentes partes del lenguaje (clases, objetos, tipos, etc.) que

nos permiten desarrollar nuestras soluciones informáticas.

Visual Studio .NET es el entorno RAD (Rapid Application Development o Desarrollo

Rápido de Aplicaciones), que nos permite utilizar .NET Framework para desarrollar nuestras

aplicaciones de una forma rápida y visual, incluyendo características de desarrollo

como el IntelliSense. Visual Studio .NET utiliza por tanto, .NET Framework.

Aspecto de Visual Studio .NET

Para desarrollar aplicaciones .NET, deberemos por tanto, utilizar necesariamente .NET

Framework, ya sea utilizando Visual Studio .NET o no.

Hablando del entorno .NET Framework, diremos que Microsoft ha añadido en él las

capacidades y características necesarias para hacer de este modelo, un modelo POO o

modelo de programación orientada a objetos.

Todos los desarrollos que realicemos con .NET serán desarrollos orientados

a objetos. Este cambio de «chip» es un cambio especialmente problemá-

6-47

Lenguajes de programación

tico para los desarrolladores que estaban acostumbrados a trabajar con Visual Basic sin

tener conocimientos sobre la orientación a objetos. La problemática llega porque será

necesario cambiar el esquema de trabajo que llevábamos a cabo cuando trabajábamos con

Visual Basic. Ahora es necesario tener claro lo que se va a hacer, cómo se va a hacer y cuándo

se va a hacer.

**16.3. ¿Todos los lenguajes para un entorno?**

Dentro del .NET Framework conviven, como hemos comentado, diferentes lenguajes

de desarrollo; sin embargo, todos comparten una serie de características que son idénticas

para cada uno de ellos.

La más importante es que comparten el mismo entorno de trabajo, el comentado

.NET Framework, el cual contiene todo lo necesario para programar, compilar y ejecutar

nuestras aplicaciones.

Alguna de las características de este entorno único es la posibilidad de compartir el

mismo CLR.

El CLR (Common Language Runtime, es decir, el Motor Común de Ejecución) es el

centro neurálgico del .NET Framework encargado de gestionar la ejecución de nuestras

aplicaciones, aplicar parámetros de seguridad y ejecutar el denominado recolector de basura

entre otras cosas. La particularidad del CLR es que tendremos uno distinto por cada plataforma,

ya sea una plataforma Windows, Linux, etc. Esto significa que una aplicación

desarrollada bajo Microsoft Windows 2000 Advanced Server, por ejemplo, debería poder

ejecutarse en un entorno que dispusiese de un CLR para Linux y un CLR para Windows.

El CLR está formado, como hemos ya indicado, por diferentes partes que son igualmente

importantes. De esta manera, nos encontramos con diferentes «cajas» que veremos

a continuación.

Cuando desarrollamos una solución en .NET Framework, la compilamos y la ejecutamos

posteriormente, pero debemos tener en cuenta diferentes aspectos que suceden de

manera transparente para el desarrollador.

Todas las aplicaciones .NET son compiladas a un lenguaje neutral denominado IL

(Intermediate Language, es decir, Lenguaje Intermedio). El CLR es capaz de compilar ese

lenguaje intermedio a lenguaje máquina, específico para cada sistema en el cual se ejecuta.

Ese es el «truco» que utiliza .NET Framework para poder ejecutar una misma aplicación

en Windows o Linux por ejemplo.

6-48

Desarrollo de Sistemas

Estructura general

de Microsoft .NET Framework

Entre otras características, el CLR contiene un mismo CTS (Common Type Specification,

es decir, Especificación de Tipos de Datos Común). El CTS, para hablar de forma

clara, constituye los diferentes tipos y definiciones de cada tipo de datos utilizable en una

aplicación .NET. Un tipo de dato no encontrado en el CTS es devuelto como error por el

CLR. Cada tipo de dato hereda su tipo del objeto o clase System.Object. Relacionado con

el CTS, nos encontramos con la CLS (Common Language Specification, es decir, la Especificación

Común de Lenguajes), la cual no es otra cosa que la especificación o reglas a

seguir a la hora de trabajar con los tipos de datos.

Pero no solo los tipos son parte fundamental de .NET Framework; también el BCL

(Base Class Library, es decir, la Biblioteca de Clases Base) es importante dentro de la estructura

de desarrollo .NET. Dentro del BCL encontraremos una extensa biblioteca formada

por clases que nos proporcionarán la posibilidad de acceder a una gran cantidad de servicios.

En el .NET Framework, referenciamos a las BCL mediante lo que se ha denominado

Namespace (Espacios de Nombres) y que se engloban dentro del Namespace System.

Además de todos los aspectos que acabamos de ver, debemos tener en cuenta otros

no menos importantes, algunos de ellos ya comentados pero no explicados. Uno de ellos

es el denominado recolector de basura o garbage collector que se incluye dentro del CLR.

El recolector de basura hace las tareas «sucias» de .NET Framework. Es el gestor de

limpieza de .NET. Su objetivo es el de eliminar de la memoria los objetos que no sean útiles

para el programador. Si bien el recolector de basura se ejecuta solo cada vez que detecta

que no hay espacio suficiente para ejecutar un objeto, podemos personalizar la ejecución

del recolector de memoria y lanzarla cuando consideremos oportuno. El programador

no debe preocuparse por los objetos, su existencia, eliminación,... en otras palabras, no

debe preocuparse por la gestión posterior de los objetos.

**16.4. .NET Framework, un entorno de desarrollo de nuestro tiempo**

Si algo es el .NET Framework es un entorno de desarrollo de nuestro tiempo, un

entorno de desarrollo moderno. En él podemos conjugar los aspectos modernos y actuales

para cubrir las necesidades de los desarrolladores. Es un entorno capaz de resolver las

necesidades de los desarrolladores de hoy, capaz incluso de trabajar con los errores que se

sucedan en los programas a modo de excepciones. Es decir, podemos trabajar con los errores

de una aplicación que se producen en tiempo de ejecución en cualquier momento. El

trabajo con excepciones nos ofrece capacidades de gestión de errores mucho mayores a las

que estábamos acostumbrados con los antiguos entornos de desarrollo de Microsoft.

Y hablando de errores, ¿quién no ha tenido alguna vez problemas con las DLL en una

aplicación Windows? .NET Framework es un entorno orientado a objetos, por lo que a la

hora de desarrollar aplicaciones, trabajaremos con objetos y clases en lugar de trabajar con

DLLs, aunque si lo deseamos, podremos seguir utilizando las DLL en nuestros desarrollos.

.NET Framework es, además, un entorno abierto. Cuando decimos abierto,

queremos decir que es un entorno adaptable o receptivo a nuevos lenguajes

de programación y tecnologías. Dentro de .NET Framework, podemos

6-49

Lenguajes de programación

hacer uso de un conjunto de lenguajes de desarrollo determinado, pero una empresa externa,

puede desarrollar su propio lenguaje de desarrollo o compilador para la plataforma

.NET, tan solo se ha de seguir unas normas para adaptarse al entorno .NET. Algunas de

estas normas constituidas dentro del CLR son las que se han expuesto con anterioridad.

De hecho, existen ya lenguajes de desarrollo para la plataforma .NET como Fortran .NET,

Cobol .NET, Phyton .NET, etc., pertenecientes a otras empresas externas.

Otra de las características de .NET Framework como entorno abierto, es que ha sido

desarrollado con la pretensión de cumplir con todos los estándares actuales, siguiendo

patrones de estandarización ya aprobados como el ECMA (http://www.ecma.ch/). Todo en

el .NET Framework, cumple con los patrones de la normalización y se apoya en estándares

abiertos. Este cambio de rumbo de Microsoft es claramente una apuesta por el desarrollador,

dándole libertad absoluta en sus decisiones y desarrollos.

Pero si algo es especialmente interesante dentro de .NET Framework es todo lo relacionado

con la seguridad. La seguridad dentro del entorno .NET, proporciona la posibilidad de

ser sensible a tipos o roles de ejecución, es decir, se puede restringir la ejecución de una aplicación

según diferentes parámetros. En relación con la seguridad, dentro de .NET Framework,

podemos trabajar con el cifrado de información según los algoritmos SHA-1 y MD5.

**16.5. .NET es a XML lo que XML es a .NET**

XML (eXtensible Markup Language, es decir, Lenguaje de Marcadores

Extensible) es un lenguaje de marcas cuya particularidad reside en

que está aceptado por el W3C, es decir, es un lenguaje estándar lo cual

significa que un sistema Windows puede entenderse con cualquier otro

sistema mediante XML y al revés.

Esta es una noticia especialmente esperanzadora para los desarrolladores, ya que no

importa la plataforma en la cual se desarrollen sus aplicaciones ni tampoco dónde se ejecuten.

Lo más importante es que el lenguaje XML es un lenguaje universal, capaz de ser

manipulado siguiendo un conjunto de reglas necesarias para que el lenguaje XML sea

correctamente interpretable.

Algunas personas definen XML como una tecnología, otros como un lenguaje y otros

ni siquiera lo definen como lenguaje ni como tecnología. XML, sin embargo, forma una

parte muy importante a tener en cuenta dentro de .NET, por lo que es conveniente tener

algunos conocimientos básicos sobre XML para poder utilizarlos en .NET sin problemas,

conociendo lo que se realiza en cada instante.

Uno de los usos más importantes de XML es el que tiene que ver con los servicios

web, más conocidos como XML Web Services (Servicios Web XML).

Otro de los usos de XML es el que se da con el trabajo de fuentes de datos junto a ADO

.NET. Sin embargo, XML se utiliza dentro de muchos ficheros de configuración

y aplicaciones dentro del propio entorno .NET. XML es una tecnología o lenguaje

que se utiliza ya en otras plataformas de desarrollo, no solo en .NET.

6-50

Desarrollo de Sistemas

**16.6. Conclusiones**

En este epígrafe hemos visto algunas de las características más destacables de la plataforma

.NET Framework. El desarrollo de aplicaciones .NET puede ser muy versátil, por lo

que conviene entender con claridad las partes fundamentales de .NET Framework. Como

ejemplos de versatilidad en el desarrollo, destacaremos la posibilidad que nos ofrece .NET

Framework, de poder desarrollar una aplicación que contenga una parte escrita en Visual

Basic .NET y otra parte escrita en C# por ejemplo. Por otro lado, en este artículo hemos

diferenciado el .NET Framework de Visual Studio .NET, el cual en algunas ocasiones, es

confundido por el desarrollador que se sienta delante de .NET por primera vez. Esperemos

que con estas explicaciones, quede claro lo que es la plataforma .NET Framework, qué partes

lo componen y cuáles son sus características más destacables.

Visual Basic .NET usa una jerarquía de clases que están incluidas en el .NET Framework,

por tanto conocer el .NET Framework nos ayudará a conocer al propio Visual Basic

.NET, aunque también es necesario conocer la forma de usar y de hacer del VB ya que,

aunque en el fondo sea lo mismo, el aspecto sintáctico es diferente para cada uno de los

lenguajes basados en .NET Framework; si no fuese así solo existiría un lenguaje.

«.NET Framework es un entorno para construir, instalar y ejecutar servicios Web y

otras aplicaciones.

Se compone de tres partes principales: el Common Language Runtime, las clases Framework

y ASP.NET»

Lo que dice la MSDN Library:

— «El .NET Framework es un entorno multi-lenguaje para la construcción, distribución

y ejecución de Servicios Webs y aplicaciones.»

— «El .NET Framework es una nueva plataforma diseñada para simplificar el desarrollo

de aplicaciones en el entorno distribuido de Internet.»

— «El .NET Framework consta de dos componentes principales: el Common Language

Runtime y la librería de clases .NET Framework.»

El Common Lenguage Runtime (CLR) es una serie de librerías dinámicas (DLLs),

también llamadas assemblies, que hacen las veces de las DLLs del API de Windows así

como las librerías runtime de Visual Basic o C++. Como sabrás, cualquier ejecutable

depende de una forma u otra de una serie de librerías, ya sea en tiempo de ejecución como

a la hora de la compilación. Pues el CLR es eso, una serie de librerías usadas en tiempo de

ejecución para que nuestros ejecutables o cualquiera basado en .NET puedan funcionar.

Por otro lado, la librería de clases de .NET Framework proporciona una jerarquía de

clases orientadas a objeto disponibles para cualquiera de los lenguajes basados en .NET,

incluido el Visual Basic. Tendrá a su disposición todas las clases disponibles

para el resto de los lenguajes basados en .NET.

VB.NET ahora es totalmente un lenguaje orientado a objetos.

6-51

Lenguajes de programación

**17. Clasificación de los lenguajes de programación**

Los lenguajes de programación constan de:

— Léxico. Conjunto finito de símbolos, a partir del cual se define el vocabulario del

lenguaje, la ortografía del lenguaje.

— Sintaxis. Conjunto finito de reglas para la construcción de las sentencias correctas

del lenguaje, la gramática del lenguaje.

— Semántica. Asociar un significado a cada posible construcción del lenguaje.

Podemos decir que un lenguaje de programación consta de un conjunto de símbolos

y un conjunto de reglas válidas para componerlos, de forma que formen un mensaje con

significado para el ordenador.

**17.1. Lenguaje máquina**

Los ordenadores solo entienden el código máquina. Este lenguaje utiliza un código

binario (0 - 1).

Las instrucciones en este lenguaje tienen dos partes diferenciadas: código de operación

y código de operando/s:

— En el código de operación se codifica la operación que realiza la instrucción. Este

código de operación siempre es único para cada instrucción.

— En el código de operando se indica la dirección binaria absoluta de memoria en

la que se encuentra el operando, con un máximo de tres, sobre el que se aplicará

la operación.

Como cada tipo de ordenador tiene su código máquina específico, para programar en

este lenguaje el programador debe conocer la arquitectura física del ordenador.

**Ventajas:**

— Los programas son directamente interpretables por el procesador central.

— Los programas no se necesitan transformaciones previas para ser ejecutado.

— Los programas se ejecutan muy eficientemente, ya que se crean específicamente

para los circuitos que lo han de interpretar y ejecutar.

— Los programas pueden utilizar la totalidad de los recursos de la máquina.

**Inconvenientes:**

— Las instrucciones son cadenas de ceros y unos (estas cadenas se pueden

introducir en el ordenador mediante un código intermedio:

octal o hexadecimal).

6-52

Desarrollo de Sistemas

— Los datos se utilizan por medio de las direcciones de memoria donde se encuentran

(binarias absolutas).

— El repertorio de instrucciones es muy reducido y las instrucciones realizan operaciones

muy simples.

— Existe poca elasticidad, flexibilidad y versatilidad para la redacción de instrucciones.

Estas tienen un formato rígido en cuanto a posición de los distintos campos

que configuran la instrucción (código de operación, dirección o direcciones de

memoria, códigos de puertos, etc.).

— El código de operación debe seleccionarse estrictamente entre los que figuran en

una tabla o repertorio fijo.

— Un programa máquina no permite el uso de sentencias declarativas, existiendo

solo las instrucciones.

— No pueden incluirse comentarios.

— Es muy difícil de reconocer o interpretar por el usuario.

— La dependencia del lenguaje máquina de la configuración de la CPU hace que

los programas redactados en este lenguaje de programación sean poco transferibles

o transportables de un ordenador a otro (no portabilidad).

El lenguaje máquina depende íntimamente a la CPU del computador. Si dos ordenadores

tienen CPU diferentes, tendrán distintos lenguajes máquina. Dos ordenadores con

el mismo microprocesador e iguales circuitos de control, tienen igual lenguaje máquina.

**17.2. Traductores**

Los lenguajes simbólicos permiten utilizar una simbología y terminología próximas a

las tradicionalmente utilizadas en la descripción de problemas.

Dado que un ordenador solo puede interpretar y ejecutar código máquina, existen

programas traductores, que traducen el lenguaje simbólico al lenguaje máquina. El código

inicial se denomina programa fuente y el programa obtenido tras el proceso de traducción

programa objeto.

Existen dos tipos de lenguajes que necesitan de un traductor: los ensambladores y los

lenguajes de alto nivel.

**17.2.1. Ensambladores**

Los lenguajes ensambladores permiten al programador:

— Escribir las instrucciones utilizando, en vez de códigos binarios o

intermedios, una notación simbólica o mnemotécnica para representar

los códigos de operación.

6-53

Lenguajes de programación

— Los códigos mnemotécnicos están constituidos por tres o cuatro letras que, en

forma abreviada, indican la operación a realizar: SUB (sustracción), MOV (movimiento),

CALL (llamada a un procedimiento), etc.

— Utilizan direcciones simbólicas de memoria en lugar de direcciones binarias absolutas.

— Existen sentencias declarativas (también denominadas pseudoinstrucciones o

directivas) para indicar al traductor la correspondencia entre direcciones simbólicas

y direcciones de memoria. Con estas pseudoinstrucciones, el traductor crea

una tabla con cuya ayuda, al generar las instrucciones máquina, sustituye las

direcciones simbólicas por las direcciones binarias correspondientes.

— Las instrucciones escritas en este lenguaje, guardan una estrecha relación con las

instrucciones del lenguaje máquina en que posteriormente serán traducidas.

— Hace corresponder a cada instrucción en ensamblador una instrucción en código

máquina.

— Incluyen líneas de comentarios entre las líneas de instrucciones.

Un programa en ensamblador no puede ejecutarse directamente por el ordenador,

siendo necesario ser traducido (ensamblado). El traductor de lenguaje ensamblador a lenguaje

máquina se denomina ensamblador. El ensamblador mejora o resuelve algunos de

los problemas de los lenguajes máquina pero siguen persistiendo otras limitaciones (repertorio

de instrucciones reducido, poca elasticidad para la redacción de instrucciones, o que

está íntimamente ligado a la CPU del ordenador).

Hay unos lenguajes evolucionados de los ensambladores, que se denominan macroensambladores.

Con ellos se solventa en cierta medida la limitación de tener un repertorio

de instrucciones muy reducido. Los lenguajes macroensambladores disponen de macroinstrucciones,

como por ejemplo transferir un bloque de datos de memoria principal a disco,

multiplicar, dividir, etc.

La macroinstrucción es una llamada a un módulo o rutina, llamada macro, que el traductor

inserta, antes de realizar el proceso de generación del código máquina definitivo, en

el lugar de la llamada correspondiente. A cada macroinstrucción le corresponden varias instrucciones

máquina y no solo una.

**17.2.2. Lenguajes de alto nivel**

Los lenguajes de alto nivel no obligan al usuario a conocer los detalles del ordenador

que utiliza. Con estos lenguajes las operaciones se expresan con sentencias o frases muy

parecidas al lenguaje matemático o al lenguaje natural.

Las características de los lenguajes de alto nivel son:

— Las instrucciones se expresan por caracteres alfabéticos, numéricos y

caracteres especiales.

6-54

Desarrollo de Sistemas

— Se pueden definir las variables que desee.

— La asignación de memoria para variables y constantes las hace directamente el

compilador.

— El repertorio de instrucciones es muy amplio, conteniendo operadores y funciones

de una gran diversidad: aritméticas, lógicas, de tratamiento de caracteres,…

— El programador puede definir sus instrucciones con una gran versatilidad, siendo

las reglas gramáticas de los lenguajes muy abiertas.

— Los lenguajes de alto nivel apenas dependen de la máquina.

— Pueden incluirse comentarios en las líneas de instrucciones, o puede haber líneas

específicas de comentarios. Esto facilita la legibilidad de los programas, tanto

para el propio programador, como para otras personas.

— Un programa escrito en un lenguaje de alto nivel no puede ser directamente

interpretado por el ordenador, siendo necesario realizar previamente su traducción

a lenguaje máquina.

Usualmente la traducción se hace en dos etapas: primero a ensamblador, y posteriormente

a código máquina. Por lo general, una sentencia en un lenguaje de alto nivel da

lugar, al ser traducida, a varias instrucciones en ensamblador o lenguaje máquina.

Entre sus actividades, el American National Standard Institute (ANSI) se encarga de

realizar normalizaciones de lenguajes para garantizar la translabilidad de los programas.

Existen dos tipos de traductores para los lenguajes de alto nivel:

**A) Compiladores**

Los compiladores traducen el código fuente a código objeto, para todo el programa a

la vez. A su vez llevan a cabo optimizaciones del programa que permiten que el programa

ocupe menos espacio o sea más rápido.

Un compilador traduce un programa fuente, escrito en un lenguaje de alto nivel, a un

programa objeto, escrito en lenguaje ensamblador o máquina. El programa fuente suele

estar contenido en fichero y el programa objeto pasa a ocupar otros ficheros. El fichero

objeto puede almacenarse en memoria masiva para ser procesado posteriormente.

La traducción por un compilador consta de dos etapas fundamentales, que a veces no

están claramente diferenciadas a lo largo del proceso:

— La etapa de análisis del programa fuente.

— La etapa de síntesis del programa objeto.

6-55

Lenguajes de programación

A su vez, cada una de estas etapas conlleva la realización de varias fases, y en cada una

de las cuales se recorre o analiza completamente el programa fuente.

**1. Análisis lexicográfico.**

Consiste en descomponer el programa fuente en sus elementos constituyentes, es decir, sus símbolos, que son caracteres o secuencias de caracteres con significado especial. El analizador léxico (también denominado escáner) aísla los símbolos, identifica su tipo y almacena en las tablas de símbolos la información del símbolo que pueda ser necesaria durante el proceso de traducción. La representación

obtenida en esta fase contiene la misma información que el programa fuente, pero de forma más compacta.

**2. Análisis** sintáctico**.**

La sintaxis de los lenguajes de programación se especifica mediante un conjunto de reglas (la gramática del lenguaje). Esta fase deberá comprobar si un programa es **sintácticamente correcto,** es decir, si sus estructuras (expresiones, sentencias o asignaciones) están construidas de acuerdo con las reglas del lenguaje.

**3. Análisis semántico.**

La semántica de un lenguaje de programación define el significado dado a las distintas construcciones **sintácticas**. En los lenguajes de programación, el significado está ligado a la estructura sintáctica de las sentencias. En el proceso de traducción, el significado de las sentencias se obtiene de la identificación

sintáctica de las construcciones sintácticas y de la información almacenada en la tabla de símbolos.

**4. Generación de código intermedio.**

Si no se han producido errores en algunas de las etapas anteriores, este módulo realiza la traducción a un código interno propio del compilador, denominado Código Intermedio, a fin de permitir la transportabilidad del lenguaje a otros ordenadores.

**5. Optimizaciones.**

En la fase de optimización se mejora el código intermedio generado anteriormente, analizando el programa de forma global.

**6. Generación de código objeto.**

En esta etapa se genera el código objeto final. En algunos casos, este código es directamente ejecutable, y en otros necesita algunos pasos previos a la ejecución (ensamblado, encuadernación y carga).

La compilación es un proceso complejo y que consume a veces un tiempo muy superior a la propia ejecución del programa. En cualquiera de las fases de análisis el compilador puede dar mensajes sobre los errores

que detecta en el programa fuente, cancelando en ocasiones la compilación para que el usuario realice en el fichero las correcciones oportunas.

Existen compiladores que permiten al usuario omitir o reducir las fases de optimización, disminuyéndose así el tiempo global de la compilación.

**B) Intérpretes**

Los intérpretes traducen el código fuente línea por línea, sin generar programa

objeto, y traduciendo las instrucciones en comandos para el hardware. Son más lentos

que los compiladores, puesto que tienen que interpretar una línea cada vez que pasan

por ella.

Un intérprete hace que un **programa fuente** escrito en un lenguaje vaya, sentencia a sentencia, traduciéndose a código objeto y sea ejecutado directamente por el ordenador.

El intérprete capta una sentencia fuente y la traduce, expandiéndola en una o varias instrucciones

máquina, que ejecuta inmediatamente, no creándose, por tanto, un fichero o **programa objeto** almacenable en memoria masiva para posteriores ejecuciones.

Características de los lenguajes interpretados:

— Las optimizaciones solo se realizan dentro del contexto de cada sentencia.

— Si una sentencia forma parte de un bucle, se traduce tantas veces como tenga

que ejecutarse el bucle, y no una sola vez como ocurriría en un compilador.

— Cada vez que utilicemos un programa tenemos que volver a traducirlo, ya que en

la traducción no se genera un fichero objeto que poder guardar en memoria

masiva y utilizarlo en cada ejecución.

Los intérpretes son preferibles a los compiladores cuando el número de veces que se

va a ejecutar el programa es muy bajo. Es más fácil desarrollar programas. Los lenguajes

intérpretes resultan más pedagógicos para aprender a programar, ya que el alumno puede

detectar y corregir más fácilmente sus errores. Los traductores-intérpretes ocupan, por lo

general, menos memoria que los compiladores.

En la actualidad, para un lenguaje dado pueden existir tanto compiladores como

intérpretes.

**C) Clasificación de los lenguajes de programación según el estilo de programación**

Antes de realizar esta clasificación hemos de pensar que un mismo lenguaje podría

estar incluido en más de un paradigma.

6-57

Lenguajes de programación

**• Lenguajes imperativos, procedimentales o procedurales**

Se basan en la asignación de valores. Usan la instrucción o sentencia de asignación

como construcción básica en la estructuras de los programas, son lenguajes orientados a

instrucciones. Se fundamentan en la utilización de variables para almacenar valores y en la

realización de operaciones con los datos almacenados.

Se caracterizan por:

— Uso intensivo de variables.

— Estructura de programas basada en instrucciones.

— Manejo frecuente de las instrucciones de asignación.

— Resolución de algoritmos por medio de estructuras de control secuenciales, alternativas

(condicionales) y repetitivas (iterativas).

— Incorporan mecanismos para el manejo de bloques.

— Gestionan la memoria de modo dinámico (en tiempo de ejecución).

Ejemplos de este paradigma son: Fortran, Pascal, C,…

a) Lenguajes de alto nivel, caracterizados por estar enfocados a la resolución de problemas

en campos de aplicación específicos y los programas escritos en ellos ser

fácilmente trasladables de uno a otro ordenador.

b) Lenguajes ensambladores y máquina, totalmente adaptados y predeterminados

por la CPU de la máquina.

**• Lenguajes declarativos.**

Son lenguajes de muy alto nivel cuya notación es muy próxima al problema real del

algoritmo que pretenten resolver. Están basados en la definición de funciones o relaciones.

No utilizan instrucciones de asignación (sus variables no almacenan valores). Son más fáciles

de utilizar pues están muy próximos al algoritmo. Se suelen denominar también lenguajes

de órdenes, ya que los programas están formados por sentencias que ordenan “qué

es lo que se quiere hacer”, no teniendo el programador que indicar al ordenador el proceso

detallado (el algoritmo) de cómo hacerlo”.

Hay dos tipos:

a) Lenguajes funcionales o aplicativos.

Los lenguajes funcionales son un tipo de lenguajes declarativos, en

los que los programas están formados por una serie de definiciones

6-58

Desarrollo de Sistemas

de funciones. No hay instrucciones, todo el programa es una función, todas las

operaciones se realizan por composición de funciones más simples.

Ejemplos de estos lenguajes son el LISP y el SCHEME.

b) Lenguajes lógicos.

Los lenguajes lógicos son el otro tipo de lenguajes declarativos, y en ellos los programas

están formados por una serie de definiciones de predicados (relaciones

entre objetos-datos). También se les denomina lenguajes de programación lógica

y el mayor exponente es el lenguaje PROLOG.

**• Lenguajes concurrentes**

Permiten la ejecución simultánea de dos o más tareas. Podrían llamarse también lenguajes

paralelos o simultáneos. Podría ser una característica del lenguaje o el resultado de

ampliar las instrucciones de un lenguaje que en sus orígenes no sea concurrente.

**• Lenguajes orientados a objetos**

Se dice que un lenguaje es orientado a objetos si soporta tipos abstractos de datos

(clases). Se basan en objetos (entes físicos de las clases), herencia, polimorfismo, abstracción

y encapsulado. Ejemplos serian JAVA y C#.