

Introducción a la programación

M. Jesús Marco Galindo
Josep Vilaplana Pastó

PID_00149895




Universitat Oberta
de Catalunya

www.uoc.edu


Índice

Introducción	5
Objetivos	6
1. Conceptos básicos de programación	7
1.1. Definiciones	7
1.2. Ejemplos	10
2. La programación como disciplina de ingeniería	12
2.1. Etapas en el desarrollo de un programa	12
2.1.1. Análisis de requerimientos. Definición del problema	13
2.1.2. Diseño del algoritmo	13
2.1.3. Implementación del programa	14
2.1.4. Pruebas	14
2.1.5. Operación, mejoras y mantenimiento	15
2.2. Conclusiones y motivación	15
3. Objetivos de la asignatura	17
3.1. Etapas del diseño de un algoritmo	17
3.1.1. Entender el problema	18
3.1.2. Plantear y planificar la solución	19
3.1.3. Formular la solución	19
3.1.4. Evaluar la corrección de la solución	19
3.2. Implementación de un programa	20
Resumen	22
Glosario	23
Bibliografía	23

Introducción

Este módulo introduce el mundo de la programación como disciplina de la ingeniería. A partir de aquí se comprenderán mejor los objetivos que hay que alcanzar para convertirse en un buen programador. En realidad, éste es, como veréis, el objetivo de esta asignatura. 

La finalidad básica de la programación es solucionar problemas mediante el ordenador. Al igual que en cualquier disciplina, el aprendizaje debe ser progresivo: primero hay que aprender a resolver problemas sencillos a partir de un conjunto de herramientas básicas, y más adelante se intentan resolver problemas más complejos que requieren más herramientas. Así pues, este material está formado por una serie de módulos con una estructura orientada al aprendizaje progresivo de los contenidos y, por lo tanto, es muy importante asimilar los contenidos de un módulo antes de pasar al siguiente.

Antes de entrar en materia tenemos que saber de qué hablamos. Para poder seguir el discurso de la asignatura es indispensable conocer, en primer lugar, el significado de los conceptos con que trabajaremos. En el primer apartado de este módulo definiremos estos conceptos básicos. 

A partir de los conceptos podremos dar un paso adelante y seguir con el segundo apartado, que nos presenta la programación como una disciplina de la ingeniería y nos muestra cómo hay que emprender el desarrollo de un programa para convertirse en un buen profesional. Pensad que la programación no es un arte, sino una técnica.

Con todo esto, ya seremos capaces de entender los objetivos de la asignatura, que se exponen en el tercer apartado.

Una vez situados y conocidos todos estos aspectos, en el módulo siguiente estaremos en condiciones de empezar el aprendizaje de las herramientas y métodos que necesitamos para desarrollar un programa.

Objetivos

Los objetivos de este módulo son los siguientes:

- 1.** Comprender los conceptos básicos de programación, algoritmo y programa.
- 2.** Conocer las etapas básicas de desarrollo de un programa.
- 3.** Entender la diferencia entre diseño e implementación, y comprender la importancia que el diseño de algoritmos tiene en la programación.
- 4.** A partir de la comprensión de los conceptos básicos, conocer cuáles son los objetivos de la asignatura.

1. Conceptos básicos de programación

Para entender en qué consiste la programación como disciplina y cuáles son los objetivos de esta asignatura, necesitamos, antes que nada, comprender los conceptos básicos que deberemos utilizar. Así pues, los definiremos de una manera sencilla y clara.

Encontraréis los objetivos de los fundamentos de la programación en el apartado 3 de este módulo.

1.1. Definiciones

Situémonos primero en uno de los conceptos más importantes con los que trabajaremos: el concepto de *algoritmo*.

Un algoritmo se define como una descripción no ambigua y precisa de las acciones que hay que realizar para resolver un problema bien definido en un tiempo finito.

Definición de algoritmo según el diccionario

Según el Diccionario de Lengua Española de la R.A.E, algoritmo es:
"Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema."

Por lo tanto, podemos pensar en un algoritmo como una receta o guión que hay que seguir para resolver un problema determinado, normalmente a partir de una información que tenemos de entrada (por ejemplo, una receta para cocinar un plato). Un algoritmo, sin embargo, es un método general para resolver todos los casos posibles del mismo problema y, por lo tanto, debe ser independiente de los datos de entrada de cualquier caso concreto.

Para comprender completamente este concepto de *algoritmo*, será preciso definir ahora los conceptos *entorno*, *acción*, *proceso* y *procesador*.

El **entorno** es el conjunto de objetos necesario para llevar a cabo una tarea determinada. El **estado del entorno** en un momento determinado es la descripción del estado de los objetos del entorno en aquel momento concreto. Un algoritmo actúa de manera que hace cambiar progresivamente el estado de su entorno.

Ejemplo de entorno y estado

En el caso de una receta de cocina, el entorno estaría conformado por los utensilios de cocina (ollas, sartenes, etc.) y los ingredientes.

El estado de las sartenes, por ejemplo, cambiará de limpias a sucias.

Una **acción** es un suceso finito en el tiempo y que tiene un efecto definido y previsto. Una acción puede actuar sobre un entorno y lo puede modificar, es decir, se parte de un estado inicial y se llega a un estado final diferente. Una **acción elemental** es una acción que el destinatario de un algoritmo entiende y sabe procesar.

Un **proceso** es la ejecución de una o varias acciones. El algoritmo expresa unas pautas que hay que seguir para llevar a cabo una tarea concreta. El encargado de llevar a cabo el proceso es el procesador. Un **procesador** es una entidad ca-

paz de comprender y ejecutar eficazmente un algoritmo. El destinatario del algoritmo es, pues, el procesador.

Un procesador puede ser una persona, una lavadora, un ordenador, etc.

Ahora que ya sabemos qué es un algoritmo, debemos decidir cómo lo expresamos. Debemos encontrar un lenguaje que nos permita realizar una descripción no ambigua y precisa de las acciones que componen nuestros algoritmos.

Denominamos **lenguaje natural** el lenguaje que normalmente utilizamos para comunicarnos. No obstante, debido a su complejidad y ambigüedad, veremos que el lenguaje natural no nos permite definir las acciones con la precisión y claridad que queremos. En realidad, si nosotros actuamos como procesadores de alguien que nos indica cómo hay que realizar una tarea, a menudo pedimos puntualizaciones y aclaraciones de qué hay que hacer.

La lengua española o la inglesa serían ejemplos de lenguajes naturales.

Necesitamos, pues, un lenguaje más reducido y preciso. En los módulos siguientes describiremos un lenguaje concreto creado *ex professo* para poder expresar cualquier algoritmo con la claridad y precisión necesarias. Lo denominaremos **lenguaje algorítmico** o **notación algorítmica**.

Lenguajes no naturales

No es la primera vez que os encontráis en la situación de tener que utilizar una nueva notación (diferente del lenguaje natural) para expresar conceptos; por ejemplo, estáis muy acostumbrados a utilizar la notación matemática (+, -, >, log, etc.).

En esta asignatura nos dedicaremos a aprender las técnicas básicas para diseñar algoritmos. Por lo tanto, necesitaremos aprender este lenguaje algorítmico que acabamos de mencionar. En el módulo “Introducción a la algorítmica”, describiremos el lenguaje algorítmico, y en los módulos siguientes aprenderemos a utilizarlo de manera progresiva, conjuntamente con las técnicas para el diseño de algoritmos. ⚠

Una vez hemos llegado a este punto, ya hemos definido la mayoría de los conceptos principales con los que trabajaremos; y prestad atención al hecho de que, curiosamente, todavía no hemos hablado de programas. ⚠

Definiremos un programa en relación con todo lo que hemos definido hasta ahora. Para hacerlo, necesitamos aclarar también qué es un ordenador o computador.

Aunque ya tenemos una idea de lo que es, un ordenador se puede definir formalmente como una máquina compuesta por circuitos electrónicos que tiene la capacidad de resolver problemas bajo el control de unas instrucciones dadas. Un ordenador está formado básicamente por un procesador, la memoria y los dispositivos de entrada y salida que permiten su comunicación con el exterior.

Ejemplos de lenguajes de programación

Algunos lenguajes de programación son Pascal, C, C++, Cobol, Fortran, VisualBasic, Java, etc.

El ordenador procesará nuestros algoritmos, pero para hacerlo es necesario que entienda nuestro lenguaje algorítmico. Será preciso, pues, transcribir nuestros algoritmos a un lenguaje de programación, es decir, a un lenguaje capaz de ser comprendido por un ordenador.

Otra definición de ordenador

En este punto, podemos definir también un ordenador como un autómata de cálculo gobernado por un programa.

Así pues, un programa es sólo la codificación de un algoritmo en un lenguaje que el ordenador entienda.

Para expresar y diseñar algoritmos, siempre utilizamos la notación algorítmica; por otro lado, el uso y la definición de lenguajes de programación están sujetos a factores de disponibilidad, tecnología actual, etc.

En esta asignatura, a medida que aprendamos a diseñar algoritmos con nuestro lenguaje algorítmico, aprenderemos también a implementarlos en un lenguaje de programación para que se puedan ejecutar en nuestro ordenador.

Básicamente, hay dos tipos de lenguajes de programación: los **lenguajes imperativos** o **lenguajes procedimentales** y los **declarativos**, que a la vez se dividen en **lenguajes funcionales** y **lenguajes lógicos**.

En la programación imperativa, los programas son secuencias de instrucciones que deben llevarse a cabo como una receta o guión para resolver un problema determinado.

En esta asignatura estudiaremos sólo la programación imperativa, que tradicionalmente ha sido la más extendida y utilizada. !

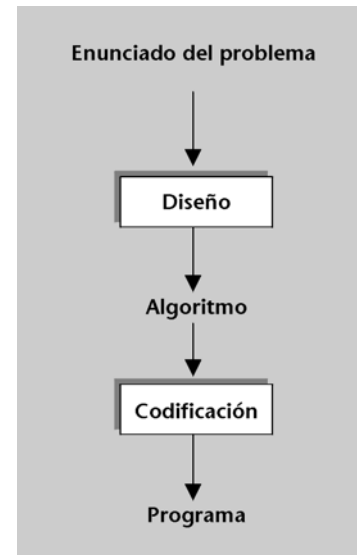
Algunos lenguajes de programación facilitan el seguimiento de una metodología concreta de programación. Una de las más ampliamente aceptada actualmente es la orientación a objetos*.

La **orientación a objetos** determina un estilo de programación que se caracteriza por el modo de manipular la información.

En asignaturas posteriores veréis el paradigma de la orientación a objetos aplicado a la programación. En esta asignatura introducimos los fundamentos básicos en los que iréis profundizando más adelante. Prestad atención al hecho de que lo realmente importante es llegar a saber diseñar un algoritmo que resuelva un problema determinado. El hecho de codificarlo para obtener el programa correspondiente consiste, simplemente, en realizar una traducción. !

El lenguaje algorítmico que definiremos y utilizaremos para diseñar algoritmos es suficientemente genérico como para que resulte bastante sencillo realizar la codificación en cualquier lenguaje de programación imperativo.

A medida que avancéis en esta materia, entenderéis más a fondo algunas de estas definiciones que puede ser que inicialmente no os queden completamente claras. Por lo tanto, conviene que repaséis continuamente este módulo.



! Los materiales correspondientes a la codificación en un lenguaje de programación los encontraréis en el aula. Con el estudio de estos materiales podréis implementar programas. Os aconsejamos que empecéis a partir de los diseños que encontraréis en estos módulos.

Ejemplos de lenguajes de programación

- a) Ejemplos de lenguajes imperativos: Pascal, Cobol y C.
- b) Ejemplos de lenguajes declarativos:
 - Lenguajes funcionales: Lisp.
 - Lenguajes lógicos: Prolog.

* Por ejemplo, C++, Object Pascal son, respectivamente, extensiones de C y Pascal que facilitan el uso de esta metodología.

! En la asignatura *Programación orientada al objeto* se explicará en qué consiste la programación bajo el paradigma de la orientación a objetos.

1.2. Ejemplos

En este subapartado veremos algunos ejemplos que nos permitirán aclarar de manera intuitiva los conceptos que acabamos de definir.

En realidad, algunos de los conceptos que hemos visto son aplicables a cualquier entorno del mundo real, no están restringidos al entorno de la informática.

Podríamos pensar en una lavadora como un autómatas, ya que es una máquina capaz de realizar una tarea de manera autónoma, en este caso, lavar la ropa.


Si pedimos a nuestra lavadora que nos lave la ropa blanca, la lavadora seguirá el proceso siguiente:

- Utilizar el agua y el jabón del cajón.
- Calentar el agua a 40 grados.
- Dar vueltas durante 20 minutos (lavar).
- Expulsar el agua.
- Tomar más agua.
- Dar vueltas durante 10 minutos (primer aclarado).
- Expulsar el agua.
- ...
- Dar vueltas durante 10 minutos (cuarto aclarado).
- Expulsar el agua.
- Dar vueltas muy rápidamente durante 1 minuto (centrifugar).

Todo este conjunto de acciones que la lavadora ha llevado a cabo sería el algoritmo que este aparato sigue para lavar la ropa blanca.

Si ahora pedimos que nos lave la ropa delicada, las acciones que ejecutará la lavadora serán seguramente diferentes (por ejemplo, la temperatura será inferior, el tiempo de lavado será más corto, habrá menos aclarados, etc.); esto equivale a decir que el algoritmo para lavar la ropa delicada es diferente del algoritmo para lavar ropa blanca.


Así pues, un algoritmo nos indica las acciones que hay que seguir para resolver un problema concreto, pero necesitamos un autómatas (procesador) que sea capaz de ejecutarlo y un entorno afectado (ropa, agua, bombo, etc.).

También es importante observar el hecho de que puede haber más de un algoritmo que resuelva el mismo problema, así como algún problema para el que no existe ningún algoritmo que lo solucione. 

Observad también que es necesario que se cumplan unas condiciones iniciales para que el problema se resuelva correctamente (que la ropa blanca esté dentro

Otros ejemplos de algoritmos

Del mismo modo que en el caso mencionado de la lavadora, también podríamos pensar en un algoritmo que resuelva el problema de cocinar una tortilla de patatas, el de cambiar una rueda de un coche, el de construir un edificio, el de preparar la declaración de la renta, el de multiplicar dos matrices, etc.

 Es importante que recordéis esto cuando hablemos de la especificación, en el módulo "Introducción a la algorítmica" de esta asignatura.

de la lavadora, que la lavadora esté conectada a la corriente eléctrica y a la entrada del agua, así como que haya jabón en el cajón). Para poder llegar al estado final deseado, que en nuestro caso sería tener la ropa limpia al final, el entorno debería estar en este estado inicial. Si no partimos del estado inicial correcto, seguramente no conseguiremos nuestro objetivo (pensad, por ejemplo, que si la lavadora está conectada a la corriente, nuestro algoritmo no dará el resultado esperado).

Volviendo a nuestro entorno, el ordenador es el autómatas que debe ser capaz de resolver los problemas que le planteemos*.

* Por ejemplo, que nos calcule la media de cuatro números enteros determinados.

Un algoritmo en el lenguaje natural para resolver el problema podría ser el siguiente: “Sumad los cuatro números que nos han dado y dividid el resultado de esta suma por cuatro para obtener el resultado final”.

Ahora veremos cómo se expresaría este algoritmo en lenguaje algorítmico. Posteriormente, podríamos implementar este algoritmo en un lenguaje de programación y ejecutarlo en nuestro ordenador.


```
algoritmo media;
  var
    n, suma, i: entero;
    resultado: real;
  fvar
    suma := 0;
    i := 1;
  mientras i ≤ 4 hacer
    leerEntero(n);
    suma := suma + n;
    i := i + 1
  fmientras
    resultado := enteroAReal(suma)/4.0;
    escribirReal(resultado)
falgoritmo
```

2. La programación como disciplina de ingeniería


Todo lo que estudiaremos en esta asignatura debe servirnos para aprender a programar correctamente. Veamos, pues, como motivación para el estudio de la programación, qué se hace en el mundo real.

Actualmente, la **programación** se considera una disciplina que, igual que otras disciplinas de ingeniería, se fundamenta en una teoría, una propuesta metodológica y un conjunto de técnicas de diseño.

Cada una de estas partes de la disciplina ha surgido de la investigación y experiencia adquiridas en los últimos años y contribuye a hacer que la programación sea una tarea eficaz (que dé los resultados esperados) y eficiente (en un tiempo de desarrollo razonable).

Evidentemente, en un entorno industrial competitivo, la **eficacia** y la **eficiencia** son valores muy apreciados en las empresas. Precisamente por este motivo la asignatura se orienta a aprender a programar de manera eficiente y a adquirir cierto grado de destreza a la hora de construir programas correctos en un tiempo razonable. No obstante, para conseguirlo necesitaréis practicar mucho. 

Para comprender mejor los objetivos concretos de la asignatura, veremos ahora a grandes rasgos cuáles son las etapas que clásicamente se siguen para desarrollar un programa.

 Encontraréis los objetivos que se especifican en el apartado 3 de este módulo didáctico.

2.1. Etapas en el desarrollo de un programa

Para obtener un programa que resuelva un problema, hay que pasar por las siguientes etapas:

1. Análisis de requerimientos. Definición del problema
2. Diseño del algoritmo
3. Implementación del programa
4. Pruebas
5. Operación, mejoras y mantenimiento

Después de la etapa de pruebas, ya tenemos un programa disponible para empezar a operar. Sin embargo, como resultado de su explotación y mantenimiento, es posible que sea necesario llevar a cabo mejoras o cambios, en definitiva, modificaciones que generalmente provocan tener que retroceder a algunas etapas previas del desarrollo.

Observación

Recordad que nos movemos dentro del paradigma de la programación imperativa. Si siguiésemos otros tipos de programación, el ciclo de desarrollo podría ser ligeramente diferente.

Por motivos de eficiencia, las empresas quieren que el desarrollo de cada una de las etapas se efectúe de manera secuencial en el tiempo y que ninguna etapa obligue a retroceder a la anterior (excepto, como ya hemos indicado, la última). Actualmente se dispone de conocimientos suficientes para que esto se pueda llevar a cabo.

En los subapartados siguientes describiremos cada una de las etapas.

2.1.1. Análisis de requerimientos. Definición del problema

Uno o varios clientes plantean un problema que necesita una solución mediante la construcción de un programa. De entrada, hay que realizar un análisis del problema, que puede ser laborioso, ya que a veces el problema es difícil de comprender. Una vez que se ha aclarado el problema, se realiza un análisis minucioso de los requerimientos que ayudarán a definir el problema que queremos resolver.


El resultado de la etapa de análisis de requerimientos y definición del problema será un enunciado preciso y claro del problema que hay que resolver.

En esta asignatura no estudiaremos esta etapa (no se puede aprender todo de golpe), ya tendréis la oportunidad más adelante en asignaturas posteriores. Por lo tanto, partiremos siempre de un enunciado ya completo, claro y concreto, y nos centraremos en las tres etapas que describiremos a continuación.


2.1.2. Diseño del algoritmo

A partir del enunciado debemos realizar un diseño que nos lleve a la solución deseada.

Progresivamente, a lo largo de cada módulo aprenderéis las técnicas, herramientas y metodologías adecuadas para poder realizar esta etapa con eficacia y eficiencia. La representación del diseño se realizará mediante el lenguaje algorítmico que se expone básicamente en el módulo “Introducción a la algorítmica” de esta asignatura.

Como veremos, ésta es una de las etapas más importantes y costosas dentro del desarrollo de un programa. Además, los objetivos de esta asignatura se centran principalmente en el aprendizaje de esta etapa. Por lo tanto, en el último apartado de este módulo veremos con más detalle en qué consiste el diseño de un algoritmo. 


2.1.3. Implementación del programa

Implementar significa *llevar a cabo*. El resultado del diseño es un algoritmo, pero, para ejecutarlo deberá traducirse el diseño realizado en lenguaje algorítmico a un lenguaje de programación que el ordenador entienda. Esta etapa es sencilla, puesto que sólo se trata de una traducción bastante mecánica del diseño ya realizado para obtener el programa correspondiente, que podremos ejecutar en el ordenador. 

Ejemplo de implementación del lenguaje

Si elegimos implementar nuestros algoritmos en C, sólo necesitaremos conocer la sintaxis y la semántica del lenguaje C, y la manera como se traduce cada operación del lenguaje algorítmico en la sentencia correspondiente en C.

Podremos traducir sin ninguna dificultad a lenguajes de programación imperativos cualquier diseño que hayamos realizado en lenguaje algorítmico. Simplemente deberéis establecer las reglas de traducción al lenguaje elegido y proceder.

 Aprenderéis la etapa de implementación en el lenguaje de programación elegido para codificar los algoritmos en la parte práctica de la asignatura. El material de esta parte práctica no está incluido en estos módulos, sino que lo encontraréis en el aula.

2.1.4. Pruebas

Finalmente, será necesario detectar posibles errores en nuestros programas.

En la etapa de pruebas se trata de probar el programa resultante con diferentes datos de entrada que reciben el nombre de juegos de pruebas. El éxito de estas pruebas dependerá en gran medida de la calidad del diseño realizado antes.

No obstante, tened en cuenta que los juegos de pruebas sólo sirven para comprobar que el programa no es correcto (si algún juego de pruebas no da los resultados esperados), pero no para asegurar que sí lo es (excepto que se comprueben todas las entradas posibles, que es prácticamente imposible).

En realidad, aunque en esta asignatura no los estudiaremos, hay métodos formales de verificación que permiten demostrar la corrección de un algoritmo –y, por lo tanto, del programa correspondiente– a partir de la especificación formal de sus requerimientos. De este modo, se podría afirmar con toda seguridad que el programa cumple sus objetivos.

De todas formas, ante la falta de verificación formal de algoritmos, en esta asignatura se dan técnicas y pautas metodológicas que pueden conducir al convencimiento de que se ha realizado un diseño correcto, y la elaboración de un juego de pruebas bueno y exhaustivo* confirmará nuestras pretensiones con una seguridad alta. Sin embargo, tened en cuenta que la seguridad en la corrección nos viene dada mediante dos etapas, la primera (diseño) es la más estratégica, y la segunda (implementación y prueba con un juego de pruebas exhaustivo) es la de confirmación.

Corrección del programa

La mejor forma de asegurar que el programa es correcto es estar convencidos de la corrección en la etapa de diseño. La metodología y las pautas de algoritmos que se proponen en los módulos siguientes os ayudarán a alcanzar la corrección del diseño.


* Por ejemplo, se pueden considerar los casos extraños o extremos.

2.1.5. Operación, mejoras y mantenimiento

Podéis pensar que, una vez obtenido y probado el programa, el proceso ya ha finalizado. Pero de la misma manera que todavía se diseñan coches nuevos, lavadoras nuevas, etc., los programas también cambian para mejorar o adaptarse a las nuevas necesidades, y por esta razón se vuelve a reproducir todo el proceso desde el principio.

Si la nueva mejora obliga a modificar partes del programa o si, por cuestiones de mantenimiento, hay que hacer correcciones, esto sólo se puede llevar a cabo con eficiencia si el diseño del programa se realiza de manera inteligible.


Un programa se escribe una vez, pero se lee muchas veces más. Conviene, pues, elaborar diseños claros, comprensibles y sencillos para que ayuden a hacer que esta etapa sea eficiente y eficaz.

El diseño también deberá estar bien documentado. Por lo tanto, será necesario aprender a documentar correctamente los programas y saber distinguir entre una documentación buena y una documentación que no aporta nada. 

2.2. Conclusiones y motivación

Las etapas anteriores tienen costes muy diferentes:

- 1) El **análisis de requerimientos** es una de las más costosas, ya que depende de muchos factores entre los que, además de los puramente técnicos, se encuentran los sociológicos y económicos.
- 2) La **etapa de diseño** es costosa y hay que dedicarle todos los esfuerzos que sean necesarios, puesto que de la calidad de ésta dependerá el coste y el éxito del resto de las etapas.
- 3) La **etapa de implementación** es actualmente la menos costosa de todo el desarrollo, puesto que, como ya hemos dicho, es una traducción bastante directa.

Conocer perfectamente un lenguaje de programación no significa saber programar. Conocer un lenguaje de programación es conocer la herramienta a partir de la que podremos implementar el diseño del programa. Así, pues, debemos esforzarnos a saber diseñar correctamente los programas, el resto será una tarea mucho menos costosa. 


- 4) El **coste de la etapa de pruebas** depende de la complejidad que tenga el programa y de que las etapas anteriores se hayan llevado a cabo correctamente.

En cualquier caso, recordad que en esta etapa no se puede estar nunca seguro del funcionamiento correcto y que la realización de un buen diseño ayudará a tener la seguridad de que el programa implementado funciona correctamen-

La importancia de la corrección

En algunas aplicaciones, como la hospitalaria o la informática de control de una central nuclear, los errores pueden tener efectos dramáticos.


te. Tal vez pensaréis que esto es un poco exagerado, pero no es así. Tened presente que los errores de un programa en funcionamiento pueden detener la producción de una gran empresa, con los graves prejuicios económicos que esto puede ocasionar.

Podríamos pensar que, cuando ya sabemos cómo se desarrolla un programa en una empresa y una vez elaborados algunos programas, ya sabemos programar, pero con esto no es suficiente. Para ser buenos programadores, capaces de afrontar problemas de gran envergadura como los que encontraremos en el mundo profesional, será necesario adquirir mucha destreza en la aplicación de los métodos y las técnicas que expondremos. Esta asignatura y las que la siguen, combinadas con la práctica continua, os permitirán alcanzar esta habilidad para ser buenos profesionales. 

Aficionados y profesionales de la programación

Alguien podría decir que sabe jugar a tenis porque alguna vez ha lanzado una pelota con una raqueta. Pero para llegar a ser un profesional de alto nivel deberá tener destreza en un conjunto de técnicas y métodos que deberá dominar para ser competitivo. En realidad, es muy poco frecuente, por no decir nada frecuente, ver a un aficionado en un partido de tenis de alto nivel. Lo mismo sucede en el mundo de la programación.

3. Objetivos de la asignatura

En esta asignatura nos centraremos básicamente en el diseño de algoritmos y su posterior implementación a un lenguaje de programación imperativo para obtener el programa correspondiente. 


Como ya sabemos...

... la etapa de diseño es una de las más costosas en el desarrollo de un programa.

A partir de un enunciado concreto, deberemos aprender a realizar el diseño de un algoritmo. Por otro lado, también hemos visto que puede haber más de un algoritmo que solucione un mismo problema. Por lo tanto, debemos tener unos criterios claros para elegir el algoritmo más adecuado en cada caso.

Un algoritmo adecuado será el que presente las siguientes características:

- a) **Corrección:** el algoritmo hace lo que realmente se pide.
- b) **Inteligibilidad:** el algoritmo debe ser claro y fácil de entender, puesto que se escribirá una sola vez, pero será necesario leerlo muchas más veces para poder mantenerlo y/o modificarlo.
- c) **Eficiencia:** el algoritmo debe llevar a cabo la tarea que se le ha encargado en un tiempo razonable.
- d) **Generalidad:** con pocos cambios, el algoritmo se debe poder adaptar a otros enunciados parecidos.

Cuando diseñéis vuestros algoritmos, tened en cuenta siempre estos cuatro criterios. La metodología que propondremos en esta asignatura os ayudará a diseñar algoritmos que los cumplan. Sin embargo, pensad también que no sólo hay que seguir la metodología, sino que, para alcanzar una cierta destreza en el diseño, es necesaria bastante práctica, puesto que tanto los conceptos como la metodología necesitan un tiempo de asimilación, y esto sólo se consigue mediante la práctica continua. 

3.1. Etapas del diseño de un algoritmo

Las etapas para diseñar un algoritmo son prácticamente las mismas que se necesitan para resolver cualquier problema de otras disciplinas. Básicamente son las siguientes:

- 1) **Entender el problema:** si no entendemos el problema, difícilmente lo podremos resolver. La especificación formal de los algoritmos es una buena herramienta para alcanzar esta etapa rigurosamente y evitar ambigüedades.

No obstante, en esta asignatura utilizaremos el lenguaje natural para especificar nuestros algoritmos. Por lo tanto, será una especificación informal y deberemos intentar expresarla sin ambigüedad y con rigor. En el algoritmo resultante, la especificación aparecerá en forma de comentarios.

La especificación de algoritmos se describe en el módulo "Introducción a la algorítmica" de esta asignatura.

2) **Plantear/planificar la solución:** la metodología propuesta (básicamente los esquemas de recorrido y búsqueda, y el análisis descendente) nos permitirá efectuar de manera eficiente esta etapa y facilitará el resto de las mismas.

Los esquemas de recorrido y búsqueda se estudian en el módulo "Tratamiento secuencial" y la técnica de diseño descendente en el módulo "Introducción a la metodología del diseño descendente" de esta asignatura.

3) **Formular la solución:** la notación o lenguaje algorítmico nos permitirá definir el algoritmo con precisión y sin ambigüedades.

El lenguaje algorítmico se describe en el módulo "Introducción a la algorítmica" de esta asignatura.

4) **Evaluar la corrección de la solución propuesta:** la inteligibilidad del algoritmo, junto con la aplicación de los esquemas, nos permitirá estar casi seguros de la corrección del algoritmo. La especificación (precondiciones, poscondiciones, etc.) del algoritmo también nos ayudará, además, a razonar sobre la corrección del algoritmo.


3.1.1. Entender el problema

Hay que asegurarse de que hemos leído correctamente el enunciado. Hay que saber qué se pide y de qué estado inicial se parte. En esta etapa se pretende ordenar ideas, identificando cuáles son las condiciones iniciales, que denominaremos **precondición**, y a partir de aquí establecer el objetivo que se quiere alcanzar, que denominaremos **poscondición**.

La especificación de algoritmos se estudia en el módulo "Introducción a la algorítmica" de esta asignatura.

Cuando definimos los objetivos y las condiciones iniciales, no podemos permitir ninguna ambigüedad. La especificación consistirá precisamente en esto.

Un **algoritmo** es una descripción clara y precisa de acciones que hay que realizar para resolver un problema bien definido en un tiempo finito. El algoritmo debe solucionar estrictamente lo que pide el enunciado, ni más ni menos. Así pues, en esta etapa hay que fijar las ideas que nos permitirán continuar en las etapas sucesivas.

No nos debemos preocupar de cómo lo haremos, sino de qué debemos hacer, qué tenemos inicialmente y qué queremos obtener. 

Ejemplos de especificación del problema

El objetivo de una lavadora es lavar la ropa que hemos introducido en la misma. Para alcanzar este objetivo es necesario que esté conectada a la corriente y al suministro de agua; pero en ningún momento hemos pensado cómo actúa la lavadora.


En el módulo "Introducción a la algorítmica" profundizaremos en este concepto y veremos que este ejemplo es bastante simple.

3.1.2. Plantear y planificar la solución

Ésta es la etapa de diseño a la que hay que dedicar más tiempo.

Lo primero que hay que hacer es ver si el problema corresponde a un tipo de problema que ya se ha estudiado en la teoría. Si es así, se aplican los esquemas conocidos para resolverlo de una manera segura y rápida.

En el módulo "Tratamiento secuencial" de esta asignatura se estudian diferentes esquemas de resolución para algunos tipos de problemas.

Tened en cuenta que si no aplicáis los esquemas teóricos de resolución, tardaréis más en encontrar la solución, ya que deberéis redescubrir por vosotros mismos la teoría y, además, no estaréis completamente seguros de la solución que proponéis. 

Para resolver problemas complejos, no será suficiente con la aplicación de esquemas, por lo que será necesario aplicar la metodología de diseño descendente. Esta metodología nos permitirá abordar problemas de mayor complejidad y convertirá el problema complejo en un conjunto de problemas menores a los que podremos aplicar lo que hayamos aprendido para problemas de menor complejidad.

La técnica de diseño descendente se ve en el módulo "Introducción a la metodología del diseño descendente", en el que sabremos en qué consiste.

3.1.3. Formular la solución

La notación algorítmica nos permite expresar de manera precisa y clara el algoritmo que proponemos.

El objetivo básico del módulo siguiente será precisamente exponeros la sintaxis y semántica de este lenguaje algorítmico.

Para hacerlo, hay que conocer perfectamente la sintaxis y la semántica de los elementos del lenguaje algorítmico propuesto. Sin este conocimiento, difícilmente podríamos proponer o expresar con precisión el algoritmo definitivo.

La formulación del algoritmo es el resultado de aplicar correctamente la metodología sugerida, y no de la combinación más o menos caprichosa de los elementos y/o construcciones del lenguaje algorítmico. Es decir, en la etapa anterior hemos pensado el diseño más allá de los elementos del lenguaje algorítmico.

3.1.4. Evaluar la corrección de la solución

Realizar el seguimiento de un algoritmo para un caso concreto de datos de entrada no nos permite asegurar que el algoritmo funciona para todos los casos.

La metodología propuesta en los módulos de esta asignatura, juntamente con la especificación informal que comentaremos en las partes del algoritmo diseñado, nos permitirá razonar sobre la corrección de este diseño.

3.2. Implementación de un programa

Una vez tengamos el diseño del algoritmo, lo codificaremos para obtener un programa. La codificación se debe introducir en el ordenador en el lenguaje de programación elegido mediante un editor que creará un fichero de texto denominado **fichero fuente**.

El texto que hemos introducido no es directamente ejecutable por el ordenador; para poderlo ejecutar, debemos convertirlo en instrucciones que el procesador pueda entender directamente. El encargado de realizar esta tarea es un programa denominado **compilador**, que traducirá el código fuente a un código que el ordenador comprenda.

Posteriormente, otro programa se encargará de efectuar el montaje y obtener un fichero que se pueda ejecutar y probar. Uno de los objetivos del **programa montador** es realizar la adaptación del código generado en el entorno del ordenador para que se pueda ejecutar.


Resumiendo, las etapas de implementación de un programa son las siguientes: traducir el algoritmo al lenguaje de programación, editar el programa, compilarlo, montarlo y ejecutarlo.

En la etapa de compilación pueden surgir errores de sintaxis del lenguaje de programación y será necesario volver a editar el programa para corregirlos. Una vez superados los errores de compilación, el montaje no debería ocasionar ningún problema.

Para la etapa de ejecución, es muy positivo diseñar un buen juego de pruebas que incluya todas las posibilidades que pensamos que el programa debe afrontar.

Cuando se producen errores en la ejecución del programa, seguramente no se ha realizado correctamente alguna de las etapas previas.


Podríamos caer en la tentación de realizar las etapas al revés o de saltarnos alguna. Es decir, editar un programa directamente, compilarlo, ejecutarlo y, si funciona, hacer el diseño. Este sistema, conocido como *ensayo y error*, crea programas complicados a causa de las modificaciones introducidas en la idea original durante cada fracaso del programa. Como consecuencia, el programa resulta poco o nada inteligible. Recordad que la programación es una dis-




Esta etapa se aprende con las prácticas del laboratorio y con la ayuda de material complementario de estos módulos. Este material os enseñará a traducir las sentencias del lenguaje algorítmico al lenguaje de programación elegido y a utilizar el software necesario (editor y compilador) para ejecutar vuestros programas.

Aparición de errores en la ejecución del programa

En la ejecución del programa pueden aparecer errores porque no nos hemos fijado bien en la traducción o en la edición del programa, o porque se haya producido alguna equivocación a la hora de planificar y plantear el diseño del algoritmo. En el último caso, deberéis estudiar muy bien el diseño de algoritmos y poner más atención en las primeras etapas de diseño. Recordad que el esfuerzo realizado y el tiempo dedicado a estas etapas hace que el resto de etapas sean mucho más sencillas y directas.

ciplina de la ingeniería y, por lo tanto, hay que seguir una metodología para no caer en la trampa del ensayo y error, y aprender a programar con el rigor necesario. 

Os acabamos de presentar los objetivos que hay que alcanzar en esta asignatura. Estaría bien que los releyséis de vez en cuando durante el curso para no perderlos nunca de vista. Más adelante también los podréis entender mejor. 

Resumen

Con los conceptos que hemos introducido hasta ahora, ya estamos en condiciones de enumerar el objetivo general de esta asignatura: aprender a diseñar algoritmos para resolver problemas de complejidad media y a implementar el programa correspondiente dentro del paradigma de la programación estructurada e imperativa.

Podríamos desglosar este objetivo en los siguientes:

- 1) Aprender a entender y especificar, aunque informalmente, los enunciados de los algoritmos.
- 2) Aprender un lenguaje algorítmico como herramienta para expresar algoritmos.
- 3) Aprender diferentes técnicas de diseño, como los esquemas, para resolver problemas sencillos, y la metodología de diseño descendente para solucionar problemas de mayor envergadura.
- 4) Aprender a utilizar las estructuras de datos adecuados a cada problema.
- 5) Aprender a combinar las herramientas y técnicas expuestas para resolver problemas.
- 6) Aprender a implementar los algoritmos en un lenguaje de programación procedimental para obtener un programa.
- 7) Aprender a utilizar el *software* necesario (editor y compilador) para poder compilar y ejecutar los programas.
- 8) Aprender a diseñar juegos de pruebas que permitan validar la corrección de los programas.

Los primeros cinco objetivos se alcanzan de manera gradual a partir del estudio de los módulos didácticos combinados con la realización de las actividades y los ejercicios propuestos. Los tres últimos, a partir de las prácticas de laboratorio y el uso del *software* y del material complementario del aula.

Glosario

algoritmo

Conjunto explícito de reglas para resolver un problema en un número finito de pasos.

codificación

Traducción de un algoritmo a un lenguaje de programación.

computador

Autómata de cálculo gobernado por un programa.

entorno

Conjunto de objetos necesarios para llevar a cabo una tarea determinada.

especificación

Formalización del enunciado de un problema.

estado

Descripción del entorno en un momento determinado.

implementación

Realización de un proyecto.

lenguaje algorítmico

Lenguaje artificial que se utiliza para expresar algoritmos.

lenguaje natural

Cualquier lenguaje que se utiliza como forma natural de comunicación: catalán, castellano, inglés, etc.

lenguaje de programación

Lenguaje creado para expresar programas y que es capaz de ser interpretado por un ordenador.

notación algorítmica

Véase *lenguaje algorítmico*.

ordenador

Véase *computador*.

programa

Codificación de un algoritmo en un lenguaje de programación que el ordenador entienda.

sintaxis

Reglas de estructura de un lenguaje.

semántica

Significado de los símbolos de un lenguaje.

Bibliografía

Botella, P.; Bofill, M.; Burgués, X.; Franch, X.; Lagonigro, R; Vancells, J. (1998). *Fundamentos de programación I*. Barcelona: Ediuoc.

Castro, J.; Cucker, F.; Messeguer, X.; Rubio, A.; Solano, L.; Valles, B. (1992). *Curs de programació*. Madrid, etc.: McGraw-Hill.

Vancells, J.; López E. (1992). *Programació: introducció a l'algorísmica*. Vic: Eumo Editorial (Tecno-Ciència).

