Capítulo 1

Conceptos de programación

Para aprender a programar es necesario tener claros ciertos conceptos generales y específicos de todo lo que envuelve el proceso de la programación. En este capítulo se trata de dar una visión general de la programación y a la vez exponer los conceptos clave para la resolución de problemas por medio de la computadora u ordenador.

1.1 Computadora u ordenador

Una computadora se puede ver como un sistema, en el que se introducen unos datos (datos de entrada), el sistema los procesa, operando con ellos y a veces transformándolos, y devuelve el resultado de ese procesamiento (datos de salida). La figura 1.1 muestra gráficamente una computadora elemental. De esta definición informal se puede deducir un modelo computacional y describirlo dando los datos de entrada que admite y las operaciones básicas que va a poder efectuar con dichos datos.

La programación consiste en escribir lo que debe hacer la computadora para resolver un problema concreto utilizando un lenguaje de programación.

1.2 Problemas, algoritmos y programas

En este apartado se introducen algunos conceptos básicos sobre los elementos que involucra la programación.

1.2.1 Problema

Un problema es una proposición encaminada a averiguar el modo de obtener un resultado, cuando se conocen ciertos datos de partida.

Los problemas se pueden dividir en tres tipos, dependiendo de las soluciones que tenga, que son:

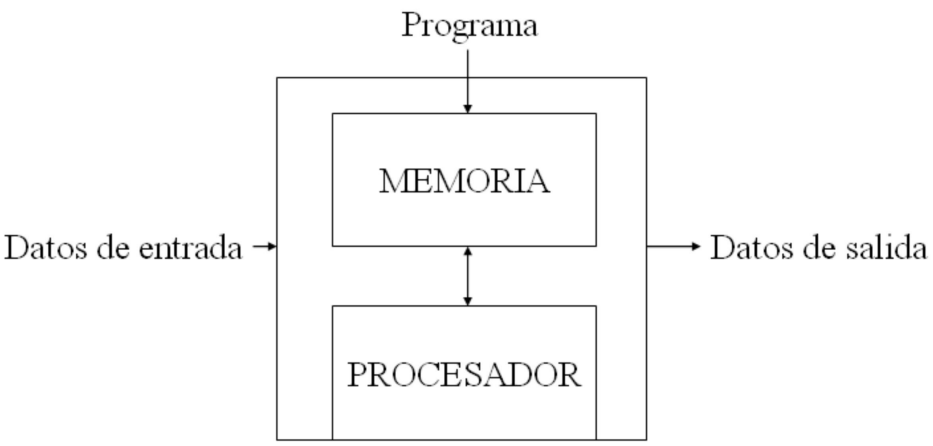

Figura 1.1: Modelo de computadora

Sin solución o irresolubles: esos problemas no pueden ser solucionados con los medios disponibles. Por ejemplo, el clásico problema de la cuadratura del círculo: "Dado un círculo, construir un cuadrado de igual área que el círculo utilizando solamente regla y compás" (la mencionada regla es no graduada y solo sirve para trazar rectas).

Determinados: los problemas tienen solución y además es única. Por ejemplo, encontrar un número natural x que resuelva la ecuación 2x=4.

Indeterminados: estos poseen un número indefinido de soluciones. Por ejemplo, hallar dos números enteros x, y que cumplen el siguiente sistema de ecuaciones: 2x-y=3; 4x-2y= -2;

Para la correcta resolución de un problema se deben seguir los pasos siguientes:

1. *Análisis del problema:* consiste en establecer con precisión qué se plantea.
2. *Especificación del problema:* consiste en dar una descripción precisa del problema con los datos de partida y el resultado. Esta descripción se puede hacer con lenguaje natural, que con frecuencia es ambiguo, o con lenguajes formales, como las matemáticas o la lógica.

1.2.2 Algoritmo

Existen varias definiciones para el concepto de algoritmo. Entre ellas destacan:

Definición 1: descripción precisa de los pasos que nos llevan a la solución de un problema planteado.

Definición 2: método tal, que partiendo de los datos apropiados, conduce sistemáticamente a los resultados requeridos en la especificación del problema.

La definición de un algoritmo describe:

- los datos de entrada,
- el proceso y operaciones que se pueden realizar y
- el resultado o datos de salida.

Propiedades de los algoritmos

Deben especificar sin ambigüedad:

- El orden de los pasos u operaciones que han de llevarse a cabo.
- Qué se realiza en cada paso.

Los algoritmos deben ser deterministas, o lo que es lo mismo, deben responder de igual modo ante las mismas condiciones y los mismos datos. Además la secuencia de pasos que se especifica debe ser finita y debe poder realizarse en tiempo finito.

Ejemplos de algoritmos son las instrucciones para ir de una ciudad a otra por carretera, cómo montar un mueble que viene en kit o una receta de cocina.

Lenguajes algorítmicos

Sirven para describir un algoritmo. Son más precisos que el lenguaje natural, pero menos rígidos (o formales) que un lenguaje de programación. Se les considera lenguajes intermedios y tienen cierta independencia de los lenguajes de programación y del computador donde se escribirá el programa. Ejemplos de lenguajes algorítmicos son el pseudocódigo, los organigramas y los diagramas de Nassi-Schneiderman (N-S) también conocidos por diagramas de Chapin. Estos dos últimos son representaciones gráficas de un algoritmo.

El *pseudocódigo* es una variación del lenguaje natural en el que se han eliminado las posibles ambigüedades mediante el uso de un vocabulario restringido y unas reglas sintácticas de construcción de sentencias. Este será el recurso utilizado para verificar y derivar programas en el capítulo 13.

Aspectos de un algoritmo

Debe cumplir una serie de características *obligatorias* como son:

- **Corrección:** respetar al pie de la letra las especificaciones del problema.
- **Complejidad:** economizar los recursos que necesita. En máquinas secuenciales, debe ahorrar tiempo y memoria, aunque la mayoría de las veces son propiedades inversamente proporcionales.

También es *deseable* que tenga las siguientes características:

- **Generalidad:** servir para la clase de problemas lo más amplia posible.
- **Eficiencia:** ser lo más eficiente posible en la medida que necesita de menos pasos para solucionar el mismo problema.

1.2.3 Problemas, algoritmos y programas

Los algoritmos indican como solucionar los problemas. Algunos problemas tienen distintas soluciones algorítmicas. Por ejemplo, hallar el máximo común divisor de dos números. Otros problemas no tienen solución algorítmica, por ejemplo el problema de la parada (encontrar un algoritmo que determine si otro algoritmo finaliza o no con unos determinados datos de entrada).

Programa: es un conjunto de instrucciones precisas, en un lenguaje "entendible" por la computadora. Hay que notar que el ordenador solo comprende directamente el lenguaje máquina.

Programación: se denomina programación a todo el proceso que conlleva la construcción de programas.

Para construir un programa hay que seguir un método. La ingeniería del software se encarga del estudio y la aplicación de los distintos métodos existentes para desarrollar completamente aplicaciones informáticas. En el apartado 1.4 se da una explicación más amplia de la ingeniería del software.

1.3 Lenguajes y paradigmas de programación

En este apartado se definen los lenguajes de programación, se muestran sus características principales y su evolución. También se da una pequeña introducción a los paradigmas y se define el usado en este libro: el paradigma imperativo.

1.3.1 Lenguajes de programación

Un lenguaje de programación es un lenguaje artificial, diseñado para representar algoritmos de forma inteligible para las computadoras.

Existen muchos lenguajes de programación, pero el único que entiende el ordenador directamente, como se ha comentado anteriormente, es el lenguaje máquina, cuyas instrucciones están codificadas en forma de secuencias de ceros y unos (bits). El resto de lenguajes necesitan traducir o interpretar las instrucciones al lenguaje máquina.

Comparación entre el lenguaje natural y los lenguajes de programación

Los lenguajes de programación son más formales y rigurosos que el lenguaje natural. Esto es debido a que sus instrucciones no son ambiguas y por lo tanto no permiten distintas interpretaciones. Todas las personas que lean una instrucción saben perfectamente qué hace (si conocen el lenguaje de programación).

Los lenguajes de programación son más simples en su sintaxis y en su semántica que los naturales. Las reglas de formación de instrucciones (sintaxis) son rígidas, al contrario que los lenguajes naturales en los que una misma frase se suele poder escribir cambiando el orden de ciertos elementos. El significado de las instrucciones viene claramente definido en los lenguajes de programación en contraposición con los lenguajes naturales en los que existe ambigüedad semántica.

Algunas características importantes de los lenguajes de programación son:

- **Sintaxis:** son reglas de formación de todas las estructuras de un programa, desde cómo construir el esqueleto de un programa entero a cómo esta formada una instrucción. Por lo tanto, especifica inequívocamente cómo están construidos los programas elaborados con un lenguaje de programación.

  Existen varias formas de especificar la sintaxis. Entre ellas destacamos las gramáticas (BNF o Backus-Naur Form) y los diagramas sintácticos.

- **Semántica:** la semántica asigna un significado a cada tipo de construcción de un lenguaje de programación. Existen diversas formas de especificación:

  - **Ejemplos (y contraejemplos)**: el significado viene dado por un ejemplo. También se suelen usar contraejemplos para especificar en caso de duda lo que no significa.
  - **Definición formal**: se usa un lenguaje formal para dar significado a la construcción.

- **Traducción y ejecución:** todos los lenguajes de programación que no son lenguaje máquina ha de traducirse al lenguaje de la máquina. Al programa escrito en un lenguaje de programación se le denomina *código o programa fuente*. Existen dos formas puras de traducción del lenguaje de programación a código máquina:

  1. **Compilación:** todo el código fuente se traduce a *código objeto*. Esta traducción la hace una aplicación especial denominada *compilador*. Por lo tanto el código objeto es el resultado del proceso de compilación. Dependiendo del compilador este código puede ser directamente ejecutable.
  2. **Interpretación:** este proceso se realiza por repetición de las siguientes fases:
     - Se traduce una instrucción del código fuente.
     - Se ejecuta dicha instrucción.

Por lo tanto, la ejecución de un programa escrito en un lenguaje de programación interpretado consiste en traducir y ejecutar una a una cada instrucción del programa fuente.

Entre las ventajas de la compilación frente a la interpretación se pueden citar las siguientes:

- Un programa compilado tarda menos en ejecutarse que uno interpretado, debido a que el proceso de traducción se ha realizado previamente.
- En la compilación se pueden hacer optimizaciones (por ejemplo, eliminar instrucciones que no se ejecutan nunca) debido a que el compilador puede ver el programa completo. En la interpretación no se realizan.
- Una vez que un programa se ha compilado y se ha generado el código ejecutable, no es necesario la presencia del compilador. En el programa interpretado, es necesario el interprete para la compilación y para la ejecución.

Los traductores de Pascal generalmente son compiladores.

- **Errores:** cuando se escribe un programa, pueden surgir errores. Dependiendo de cuando aparezcan se distinguen dos tipos:

  Errores de compilación: aparecen a la hora de compilar o interpretar el código fuente. Estos errores son fáciles de corregir debido a que generalmente los traductores dan indicaciones del tipo y el lugar del error. Ejemplos de este tipo de errores son: los sintácticos (como escribir mal una instrucción), los errores de tipo (por ejemplo intentar asignar un valor de un tipo de dato a una variable de tipo distinto), etc.

  Errores de ejecución: surgen al ejecutar el código ejecutable. Son difíciles de detectar y hacen que el programa termine de una forma no prevista. Ejemplos de este tipo de error son la realización de operaciones ilegales (división por cero), errores lógicos, etc.

1.3.2 Evolución de los lenguajes de programación

El lenguaje de programación más cercano a la computadora es el lenguaje o código máquina. En este lenguaje las instrucciones y los datos están codificadas en sistema binario y no es necesario la traducción. Entre las características del lenguaje máquina caben destacar:

- Es dependiente de los recursos de la computadora, por lo tanto, para programar, el programador debe conocer la arquitectura sobre la que programa.
- El programador se encarga de verificar que no existen errores sintácticos, pues no existe compilador.
- El programador trabaja directamente con direcciones de memoria.

Como evolución natural surgió un nuevo lenguaje llamado ensamblador, donde a cada secuencia de ceros y unos se le asocia un nombre nemotécnico. Estos nombres necesitan traducción, que se realiza mediante un programa que se llama como el lenguaje: ensamblador. Aunque fue un gran avance, todavía es necesario conocer cómo está constituida y qué recursos tiene la computadora.

Más tarde se fueron asociando nombres a conjuntos de instrucciones que realizaban una tarea compleja determinada, y a programar de manera independiente a la computadora donde se iba a ejecutar el código. A los lenguajes que incluyen estas características se les denomina lenguajes de alto nivel, porque se encuentran más cercano a la forma de pensar de los humanos que al lenguaje que entiende la máquina.

Los lenguajes de programación han ido evolucionando gracias a cuatro causas o motores que impulsan esta evolución, que son:

Abstracción: proceso mental por el que el ser humano extrae las características esenciales de algo, e ignora los detalles superfluos. Es esencial para modelar el mundo real. En un principio se hacían programas pensando como una computadora. En la actualidad se solucionan los problemas sin conocer la máquina donde va a ser ejecutado el programa.

Encapsulación: proceso por el que se ocultan los detalles de las características de una abstracción. En programación es esencial para reutilizar código. Si se ocultan los detalles de cómo está hecho un programa pero se conoce el modo de funcionamiento, se puede utilizar en cualquier otro programa sin más que respetar su especificación.

Modularidad: proceso de descomposición de un sistema en un conjunto de elementos poco acoplados (independientes) y cohesivos (con significado propio). Es esencial para abordar la resolución de problemas extensos o complicados.

Jerarquía: proceso de estructuración por el que se organizan un conjunto de elementos en distintos niveles, atendiendo a determinados criterios (responsabilidad, composición, etc.)

A medida que se fueron añadiendo estas características fueron surgiendo los siguientes estilos de programación:

Programación estructurada: se añade a la programación en código máquina la abstracción de datos e instrucciones.

Programación modular: se añade a la programación estructurada la modularidad.

Programación con tipos abstractos de datos (TAD): se añade a la programación modular la encapsulación.

Programación orientada a objetos: es la más completa, pues reúne todas las propiedades que impulsan la evolución de los lenguajes.

Las ventajas de añadir estas propiedades a los lenguajes de programación son:

- Mejor comprensión del programa.
- Mejor legibilidad.
- Mayor facilidad en el mantenimiento del programa.
- Disminución de los costes de desarrollo de una aplicación informática.

1.3.3 Paradigmas de programación

Definición: Los *paradigmas de programación* son una colección de patrones conceptuales que moldean la forma de razonar sobre problemas, de formular algoritmos y, a la larga, de estructurar programas.

Existen varias formas (paradigmas) de enfrentarse a los problemas. En este libro se va a seguir el paradigma imperativo, utilizando programación estructurada y modular. En la figura 1.2 se muestra un gráfico ilustrativo de la relación entre los paradigmas y los lenguajes de programación.

La programación imperativa o de flujo de datos se basa en el modelo Von Neumann de un computador, donde la máquina es de propósito general (permite realizar cualquier tarea computacional) y el programa que contiene las órdenes o instrucciones se encuentra almacenado en memoria. Este programa contiene un conjunto de operaciones primitivas, que se van ejecutando de manera secuencial, es decir, una detrás de otra, en el orden en que están escritas. Las variables, expresiones e instrucciones son abstracciones de las tareas y los datos que se usan. Programar en este paradigma consiste en declarar los datos (variables) necesarios y diseñar una secuencia adecuada de instrucciones (asignaciones) controlando el flujo mediante instrucciones de control. Ejemplos de lenguajes de programación que soportan este paradigma imperativo son: Pascal, C, Ada, Modula2...

1.4 Ingeniería del software

El desarrollo de programas se realiza siguiendo unos métodos. La ingeniería del software se encarga del estudio de todo el proceso que envuelve la realización de las aplicaciones informáticas y se puede definir como sigue:

Definición-1 (Bauer, 1969): el establecimiento y uso de principios robustos de la ingeniería a fin de obtener económicamente software que sea fiable y que funcione eficientemente sobre máquinas reales.

Definición-2 (IEEE, 1993): La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software.

Ambas definiciones se basan en la aplicación de métodos de ingeniería para el desarrollo de programas. Existen varios métodos pero nosotros nos fijaremos y seguiremos el denominado ciclo de vida del software que consiste en las siguientes fases o pasos:

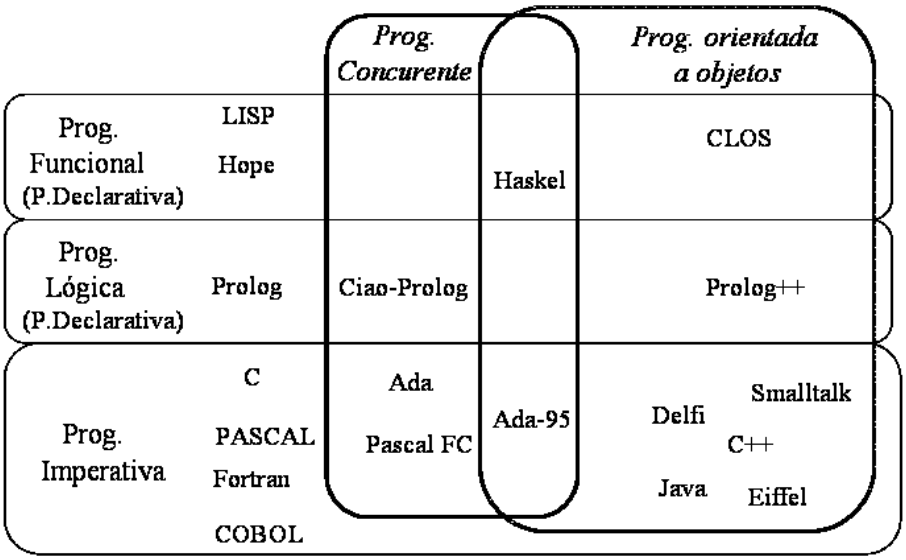

Figura 1.2: Paradigmas y lenguajes de programación

Planificación

Planificar es determinar las necesidades de programación, estimar la cantidad de recursos ya sean técnicos como humanos necesarios para el desarrollo, predecir de manera aproximada el coste y el tiempo que llevaría realizarlo y por último, determinar si el desarrollo del software es viable económicamente.

Análisis de requisitos

En esta fase se definen detalladamente las funciones de cada módulo, de acuerdo con los deseos del cliente, el trabajo conjunto de los distintos módulos, así como se establecen los criterios y sistemas de validación para comprobar que se han cumplido los intereses del cliente. En este paso también se redactan las especificaciones detalladas del funcionamiento general del software.

Diseño

Aquí se diseña el conjunto de bloques o módulos en los que se ha dividido la aplicación, y a su vez se dividen en partes o tareas que se asignan a equipos de trabajo, que posteriormente las desarrollarán y probarán independientemente del resto.

Codificación

En este paso se escriben los algoritmos de los distintos módulos en el lenguaje de programación elegido. Una vez que se han implementado se integran las partes para que formen un programa completo.

Validación

Consiste en aplicar el sistema de pruebas descrito en la fase de análisis de requerimientos. Los métodos de validación son pruebas, inspecciones y la verificación formal. Estos métodos se han de aplicar a todos los objetos de validación que son: los módulos de programa, las conexiones entre ellos (integración) y finalmente a la aplicación entera.

Mantenimiento

Aquí se redacta la documentación actualizada de todos los pasos. Se inicia la explotación del software, poniéndolo en funcionamiento real. Se detectan y subsanan errores cometidos en etapas anteriores y si es necesario se adapta la aplicación a nuevos requisitos.