2

Como programadores, desarrollaremos soluciones a los problemas, soluciones que transforman los datos dados en los resultados requeridos. Implementaremos estas soluciones en una computadora utilizando un lenguaje de programación. En este capítulo examinaremos algunas de las estrategias que se pueden utilizar para resolver problemas. Luego comenzaremos a examinar las reglas y símbolos que forman el lenguaje de programación Pascal.

PROCESO DE RESOLUCION DE PROBLEMAS

Resolvemos problemas todos los días, pero normalmente ignoramos el proceso que estamos siguiendo. En un entorno de aprendizaje, usualmente estamos dando más información de la que necesitamos: una definición clara del problema, la entrada dada y la salida requerida. En la vida real éste no es siempre el caso; frecuentemente debemos llegar nosotros mismos a la definición del problema, decidir lo que tenemos que hacer para trabajar con él y el resultado que debe dar.

Después de que hayamos comprendido y analizado el problema, debemos llegar a una solución, un algoritmo. En el Capítulo 1 hemos definido un algoritmo como un procedimiento paso a paso para resolver un problema en una cantidad finita de tiempo. Aunque siempre trabajamos con algoritmos, la mayoría de nuestras experiencias con ellos consiste en seguirlos. Seguimos una receta, practicamos un juego, ensamblamos un juguete, tomamos medicina. Todo lo que estamos enseñando es a cómo seguir directrices, es decir, a ejecutar un algoritmo.

En el Capítulo 1 describimos el proceso de programación como compuesto de una fase de resolución del problema (análisis, solución general, prueba) y una fase de implementación (solución específica, prueba, uso). En la fase de resolución del problema de la programación de computadora diseñaríamos algoritmos, no los

seguiremos. Para un problema dado se nos pedirá obtener el algoritmo, diseñar el conjunto de pasos que hay que realizar en orden a resolver el problema. Realmente, hacemos este tipo de resolución de problema siempre a un nivel inconsciente. Sin embargo, no escribimos nuestras soluciones, las ejecutamos.

Para aprender a programar tendremos que hacer conscientemente algunas de las estrategias subyacentes a la resolución de problemas, en orden a aplicarlos a los problemas de programación. Veamos algunas de las estrategias que usamos diariamente.

Hacer preguntas

Si le dan verbalmente la tarea, usted hace preguntas hasta que tiene claro lo que ha de hacer. Usted pregunta cuándo, por qué, dónde, hasta que su tarea está completamente especificada. Si sus instrucciones están escritas, puede poner preguntas en los márgenes, subrayar una palabra, agrupar palabras o sentencias, o cualquier otra forma de indicar que la tarea no está clara. Quizá sus preguntas se respondan un párrafo más adelante, o puede que tenga que discutirlas con la persona que le ha dado la tarea.

Algunas preguntas típicas que tendrá que hacer en un contexto de programación son las siguientes:

- ¿Qué me han dado para trabajar, es decir, cuáles son mis datos?
- ¿Cuál es la apariencia de los datos?
- ¿Cuántos datos hay?
- ¿Cómo sabré cuándo tendré procesados todos los datos?
- ¿Cuál debe ser la apariencia de mi salida?
- ¿Cuántas veces debo repetir el proceso que estoy haciendo?
- ¿Qué condiciones especiales de error pueden aparecer?

Buscar cosas que le sean familiares

Nunca debemos reinventar la rueda. Si existe una solución, úsela. Si hemos resuelto antes el mismo problema o uno parecido, solamente repetimos la solución. No pensamos conscientemente, "he visto esto antes y sé cómo hacerlo", nosotros la hacemos. Los humanos son buenos para reconocer situaciones similares. No tenemos que aprender a cómo ir al mercado a comprar leche, luego a comprar huevos, luego a comprar azúcar. Sabemos que ir al mercado será siempre lo mismo y lo único que varía es lo que se compra.

En informática verá ciertos problemas muchas veces bajo diferentes aspectos. Un buen programador verá inmediatamente una subtarea que ha sido resuelta antes y la conectará a la solución. Por ejemplo, encontrar la temperatura diaria más alta y más baja es exactamente el mismo problema que encontrar la mayor y la menor nota en un examen. Lo que se quiere son los números mayor y menor entre un conjunto de números (ver Fig. 2-1).

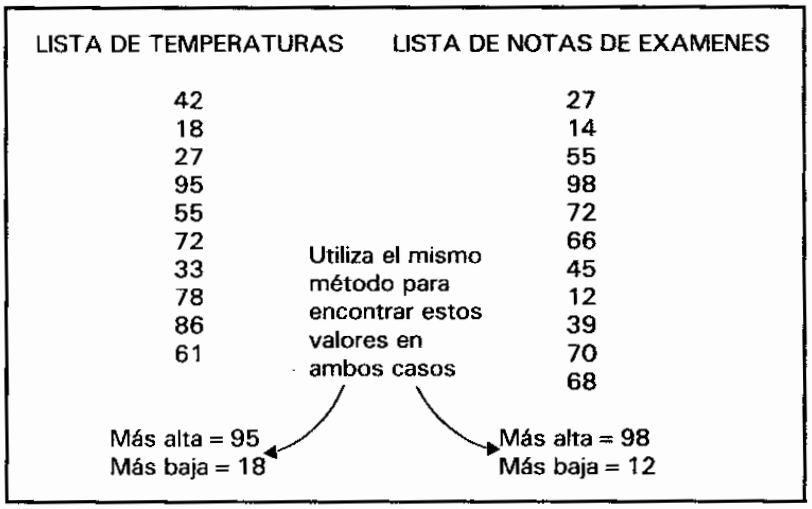

Figura 2-1. Buscar cosas familiares.

Divide y vencerás

Normalmente dividimos un problema grande en unidades más pequeñas que podamos manejar. La tarea de limpiar la casa o el apartamento puede parecer abrumadora. La tarea compuesta de limpiar el salón, el comedor, la cocina, los dormitorios y los baños parece más manejable. El mismo principio se aplica en la programación. Dividimos un gran programa en partes más pequeñas, las cuales pueden resolverse individualmente. De hecho, la metodología que estudiaremos en el Capítulo 3 para diseñar algoritmos se basa en este principio (ver Fig. 2-2).

Resolución por analogía

Frecuentemente, cuando se nos presenta un problema, intentamos recordar algún problema similar que hayamos visto antes. Los dos problemas pueden estar en

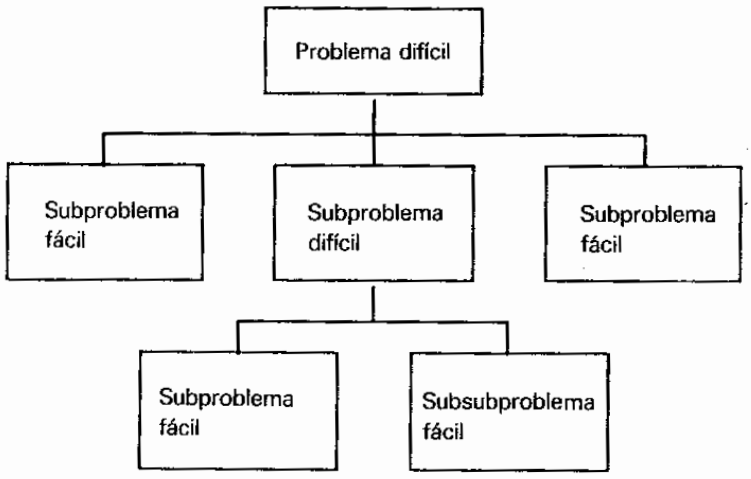

Figura 2-2. Divide y vencerás.

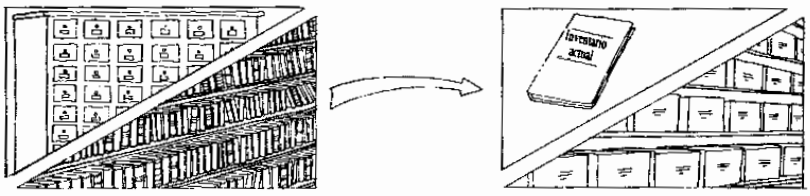

Figura 2-3.
Analogía.

Un catálogo de biblioteca puede dar algunas ideas sobre cómo organizar un inventario de piezas.

áreas completamente diferentes. Por ejemplo, algo que una vez tuvo que resolver en una clase de química puede venir a su cabeza cuando se le presenta un problema de estadística comercial. Puede tener una mayor intuición en la resolución del problema que ha de manejar si recuerda cómo resolvió el otro problema. En otras palabras, haciendo una analogía entre los dos problemas (ver Fig. 2-3).

La analogía puede ser una forma poderosa de atacar un problema. Puede inspirar un plan de acción cuando se está buscando ideas. Cuando se trabaja en un nuevo problema, se encontrarán cosas que son diferentes de las del problema viejo, pero éstos son normalmente los detalles que pueden ser fácilmente tratados uno a uno. Cuando llegue la hora de tratar con todas las "pequeñas" diferencias, descubrirá que la solución que ha alcanzado no se parece mucho con la solución con la que comenzó a trabajar. No se preocupe si la analogía no coincide perfectamente; la única razón para empezar con una analogía es que da un punto de comienzo necesario.

Los mejores programadores son las personas que tienen una gran experiencia en resolución de problemas de diferentes tipos.

El enfoque de construcción de bloques

Una forma de atacar grandes problemas es ver si existe alguna solución para pequeños trozos del problema. Puede ser posible poner alguno de éstos juntos para resolver un problema mayor. Se examina un gran problema y se ve si puede ser dividido en subproblemas que se correspondan con problemas más pequeños para los cuales existen ya soluciones. La resolución del problema mayor es entonces simplemente un asunto relativo a poner juntas soluciones existentes, como acoplar los bloques que forman una pared (ver Fig. 2-4).

Análisis medios-fines

Muchos problemas pueden caracterizarse teniendo un estado de comienzo y un estado final deseado, con un conjunto de acciones que pueden ser usadas para ir de un estado a otro. Por ejemplo, si se quiere ir de Boston, Massachusetts, a Austin, Texas, ha de conocerse que el estado de comienzo consiste en estar en Boston y que el estado final deseado es estar en Austin. El conjunto de acciones que se

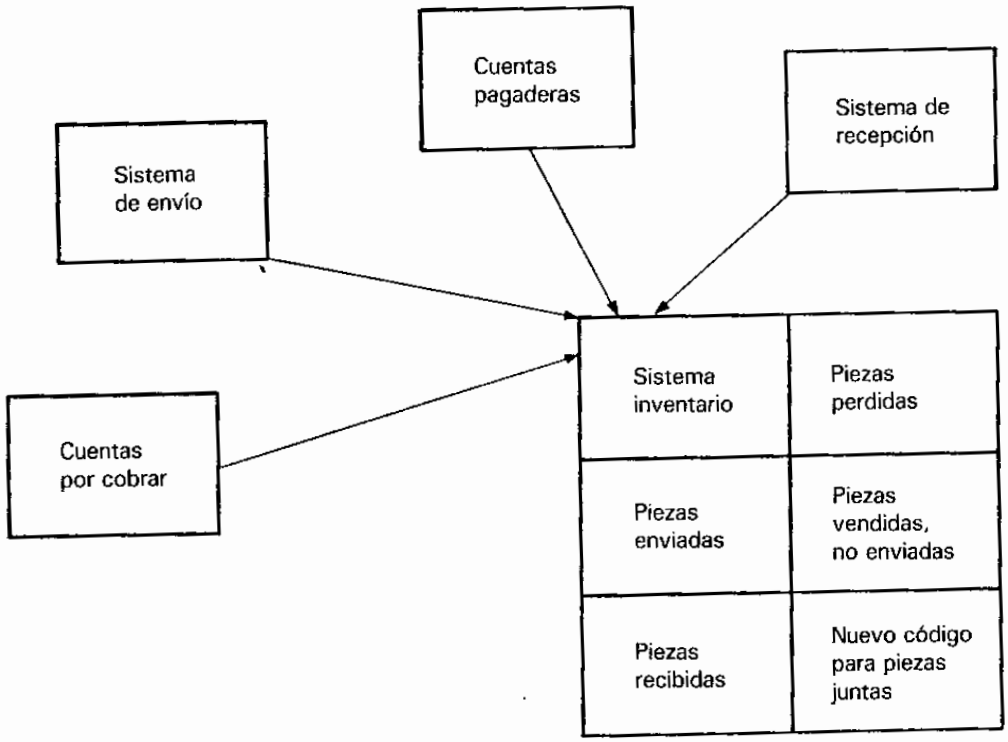

Figura 2-4. *Método de construcción de bloques.*

podría realizar para ejecutar esto es muy grande. Se podría ir conduciendo, se podría hacer autostop, se podría ir en bicicleta, se podría tomar un bote para la mayor parte del camino o se podría ir volando en un avión. El método elegido dependerá de las circunstancias asociadas al problema. Si se tiene prisa, probablemente se irá volando. Si no se tiene dinero, se consideraría ir haciendo autostop.

Una vez que hayamos elegido algún conjunto de acciones, por ejemplo ir volando, aún tendremos que resolver más detalles. Pueden existir varias rutas entre las que hemos de elegir una (¿ir por Atlanta o Chicago?). Podríamos establecer un objetivo intermedio entre los dos estados inicial y final. Quizá existiera un vuelo directo realmente barato de Newark. El objetivo intermedio podría entonces ser

Figura 2-5. *Análisis Medios-Fines.*

Partida: Boston Destino: Austin	Medios: *Volar,* andar, hacer autostop, bicicleta, conducir, barco, autobús
Partida: Boston Destino: Austin	Medios modificados: Volar a Chicago, luego Austin; *volar a Newark, luego Austin;* volar a Atlanta, luego Austin
Partida: Boston Destino: Austin Objetivo intermedio: Newark	Medios: *Vuelo diario,* andar, hacer autostop, bicicleta, conducir, barco, autobús
Solución: Tomar un vuelo diario a Newark y luego coger un vuelo barato a Austin	

llegar a Newark desde Boston. Ahora deberíamos examinar qué medios tenemos disponibles para alcanzar dicho objetivo intermedio (ver Fig. 2-5).

La estrategia global de análisis medios-fines consiste en poner los fines que han de conseguirse y luego analizar qué medios se tienen para poder llegar a ellos. En el desarrollo de una solución que se convierta en un programa de computadora, primero hay que escribir cómo ha de ser la entrada y cómo será la salida deseada. Luego se considerarán las acciones que la computadora puede ejecutar y se comienza el trabajo con una serie de dichas acciones que transformarán los datos en resultados. Puede ayudarse estableciendo objetivos intermedios. Por ejemplo, el Programa Nómina del Capítulo 1, puede decidir primero calcular los salarios y luego considerar las horas extras. Tomando estas decisiones, se establecen objetivos intermedios que son más fáciles de resolver que el objetivo final global de imprimir la nómina. Este método es algo diferente del método de divide y vencerás.

Bloqueos mentales: el miedo a comenzar

Los escritores están demasiado familiarizados con la experiencia de empezar una página en blanco, sin saber dónde comenzar. Los programadores también encuentran un bloqueo mental asociado con el primer encuentro con un problema. Cuando se examina un problema difícil parece que nos superará.

Pero siempre existe una forma de comenzar a resolver cualquier problema: escribirlo en papel con nuestras propias palabras, de forma que se comprenda bien. Una vez que se intenta parafrasear el problema, se puede enfocar cada subparte individualmente en vez de intentar comprender el problema entero de una vez. Haciendo esto, automáticamente comenzaremos a organizar las subpartes para formar un cuadro más claro del problema global. Comenzaremos a ver piezas del problema que resultan familiares o que son análogas a otros problemas que hemos resuelto. Observaremos partes que no están claras y tendremos que hacer más preguntas a las personas que nos plantearon el problema.

Al escribir el problema se intentará naturalmente agrupar cosas juntas en trozos pequeños, comprensibles, los cuales pueden ser formas naturales de dividir el problema por el método de divide y vencerás. La descripción del problema puede coleccionar toda la información sobre los datos y los resultados en una forma fácil de referenciar. Luego podremos ver los estados de comienzo y final y quizás aplicar el análisis medios-fines.

La mayoría de los bloqueos mentales son causados por no comprender realmente bien el problema. La reescritura del problema con nuestras propias palabras es una buena forma de realizar por nosotros mismos un enfoque de las subpartes del problema, una cada vez, y conseguir ver qué es lo que se necesita para llegar a la solución.

Ahora apliquemos estas estrategias (llamadas *heurísticas*) a un problema específico.

Problema

¿Cómo puede ir a la fiesta?

Preguntas

¿Dónde es la fiesta?
¿Desde dónde iré?
¿Qué tiempo hace (o probablemente hará)?
¿Iré andando? ¿Conduciendo un coche? ¿Tomaré el autobús?

Una vez se haya respondido a estas preguntas, puede empezar a diseñar su algoritmo.

Si está lloviendo, su coche está en el taller y los autobuses están parados, su mejor solución (algoritmo) puede ser llamar a un taxi y dar al conductor la dirección.

Si usted mira un mapa y ve que a donde tiene que ir está seis bloques al este del edificio dónde trabaja, la primera parte de su algoritmo puede ser repetir lo que hace cada mañana para ir al trabajo (suponiendo que va desde casa). La siguiente parte podría ser girar al este y andar seis bloques. Si tiene problemas para recordar cuántos bloques ha andado, puede tomar un lápiz y hacer una marca en un trozo de papel cada vez que cruza una calle.

Aunque hacer marcas puede ser una excepción en un algoritmo humano, es una técnica que usará frecuentemente en sus programas. Si quiere repetir un proceso diez veces, tendrá que escribir las instrucciones para contar cada vez que hace el proceso y comprobar cuándo se alcanza el diez. Esta es la construcción de repetición (bucle) mencionada en el Capítulo 1.

Si quiere escribir un conjunto de dirección para otras gentes, algunas de las cuales deben salir de su sitio y otras de otros, debe tener dos conjuntos de instrucciones precedidas de una pregunta. Si viene del lugar A, siga el primer conjunto de direcciones; si no, siga el segundo conjunto de direcciones. Este podría ser un ejemplo de la construcción selección referida en el Capítulo 1.

Llegar a un procedimiento paso a paso para resolver un problema particular no es siempre seguro y preciso. De hecho, normalmente es un proceso de prueba y error que necesita varios intentos y refinamientos. Cada intento se prueba para ver si realmente resuelve el problema. Si lo hace, bien. Si no lo hace, se intenta de nuevo.

Al diseñar algoritmos para los programas de las computadoras, es importante tener en cuenta que la computadora manipula datos —esa información que hemos reducido a una forma simbólica—. Hemos considerado algunos algoritmos que necesitan que alguien realice acciones físicas. Estos algoritmos no son posibles de usar en una computadora. Nuestro principal interés, entonces, es conocer cómo la computadora puede transformar, manipular, calcular o procesar los datos de entrada para producir la salida o resultados deseados. Podemos analizar el contenido (de qué están compuestos) y la forma (en qué orden están) de los datos de entrada, así como el contenido y la forma requerida de la salida para ayudarnos a desarrollar un algoritmo para procesar los datos.

El Programa Nómina del Capítulo 1 estaba codificado a partir de un algoritmo que describía cómo se realizaba la nómina a mano. El algoritmo para hacer algo a mano frecuentemente puede usarse con alguna o una pequeña modificación como nuestra solución general. Sólo hay que tener en cuenta las cosas que una computa-

dora puede hacer. Cuando se escribe un algoritmo para un programa, han de tenerse en mente las instrucciones permitidas en un lenguaje de programación. Este conocimiento evita diseñar algoritmos que podrían ser difíciles o imposibles de codificar.

Cuando haya comprobado a mano su algoritmo y crea que su solución es correcta, puede proceder a traducir su algoritmo a un lenguaje de programación. Esta fase de implementación es similar a la fase de resolución del problema en la que el programa (algoritmo) debe probarse con los datos de entrada para ver si produce la salida deseada. Si no lo hace, debe localizar los errores que existan en el programa. Si su algoritmo es incorrecto, debe volver a la fase de resolución del problema para ver dónde se equivocó. Algunas veces puede tener un algoritmo correcto, pero haberse equivocado al traducirlo a un lenguaje de programación.