2.2. PROGRAMACIÓN MODULAR

La *programación modular* es uno de los métodos de diseño más flexible y potente para mejorar la productividad de un programa. En programación modular el programa se divide en *módulos* (partes independientes), cada uno de las cuales ejecuta una única actividad o tarea y se codifican independientemente de otros módulos. Cada uno de estos módulos se analiza, codifica y pone a punto por separado. Cada programa contiene un módulo denominado *programa principal* que controla todo lo que sucede; se transfiere el control a *submódulos* (posteriormente se denominarán *subprogramas*), de modo que ellos puedan ejecutar sus funciones; sin embargo, cada submódulo devuelve el control al módulo principal cuando se haya completado su tarea. Si la tarea asignada a cada submódulo es demasiado compleja, éste deberá romperse en otros módulos más pequeños. El proceso sucesivo de subdivisión de módulos continúa hasta que cada módulo tenga solamente una tarea específica que ejecutar. Esta tarea puede ser *entrada, salida, manipulación de datos, control de otros módulos* o alguna *combinación de éstos*. Un módulo puede transferir temporalmente (*bifurcar*) el control a otro módulo; sin embargo, cada módulo debe eventualmente devolver el control al módulo del cual se recibe originalmente el control.

Los módulos son independientes en el sentido en que ningún módulo puede tener acceso directo a cualquier otro módulo excepto el módulo al que llama y sus propios submódulos. Sin embargo, los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo cuando se transfiera a ellos el control.

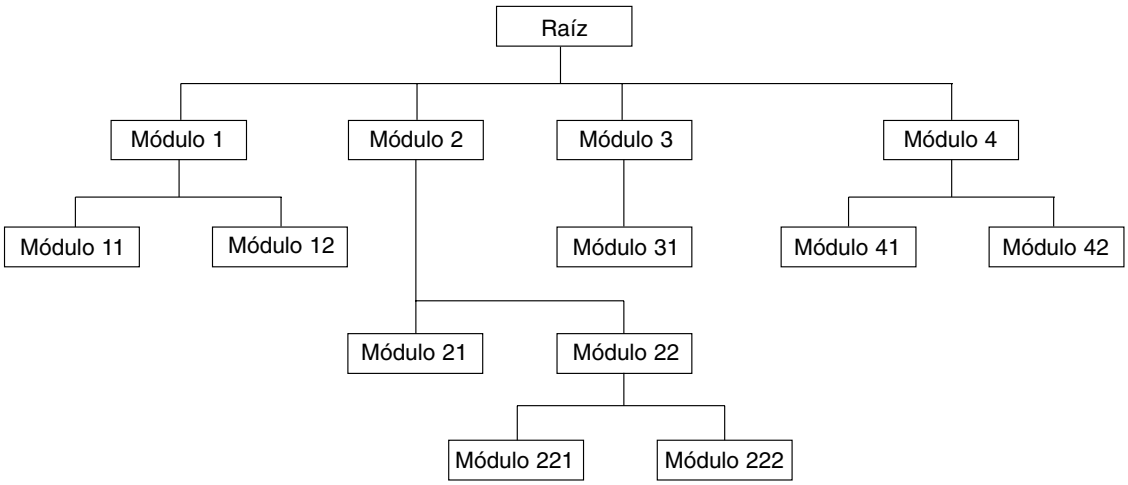


Figura 2.6. Programación modular.

Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes del mismo programa. Esto reducirá el tiempo del diseño del algoritmo y posterior codificación del programa. Además, un módulo se puede modificar radicalmente sin afectar a otros módulos, incluso sin alterar su función principal.

La descomposición de un programa en módulos independientes más simples se conoce también como el método de **«divide y vencerás»** (*divide and conquer*). Se diseña cada módulo con independencia de los demás, y siguiendo un método ascendente o descendente se llegará hasta la descomposición final del problema en módulos en forma jerárquica.

2.3. PROGRAMACIÓN ESTRUCTURADA

Los términos *programación modular, programación descendente* y *programación estructurada* se introdujeron en la segunda mitad de la década de los sesenta y a menudo se utilizan como sinónimos aunque no significan lo mismo. La programación modular y descendente ya se ha examinado anteriormente. La *programación estructurada* significa escribir un programa de acuerdo a las siguientes reglas:

- El programa tiene un diseño modular.
- Los módulos son diseñados de modo descendente.
- Cada módulo se codifica utilizando las tres estructuras de control básicas: *secuencia, selección* y *repetición*.

Si está familiarizado con lenguajes como BASIC, Pascal, FORTRAN o C, la programación estructurada significa también *programación sin* /**GOTO/** (C no requiere el uso de la sentencia **GOTO**).

El término *programación estructurada* se refiere a un conjunto de técnicas que han ido evolucionando desde los primeros trabajos de Edgar Dijkstra. Estas técnicas aumentan considerablemente la productividad del programa reduciendo en elevado grado el tiempo requerido para escribir, verificar, depurar y mantener los programas. La programación estructurada utiliza un número limitado de estructuras de control que minimizan la complejidad de los programas y, por consiguiente, reducen los errores; hace los programas más fáciles de escribir, verificar, leer y mantener. Los programas deben estar dotados de una estructura.

La **programación estructurada** es el conjunto de técnicas que incorporan:

- *recursos abstractos,*
- *diseño descendente (top-down),*
- *estructuras básicas.*

2.3.1. Recursos abstractos

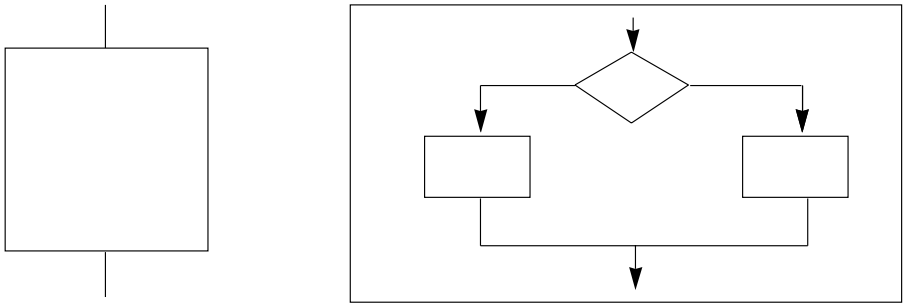
La programación estructurada se auxilia de los recursos abstractos en lugar de los recursos concretos de que dispone un determinado lenguaje de programación.

Descomponer un programa en términos de recursos abstractos —según Dijkstra— consiste en descomponer una determinada acción compleja en términos de un número de acciones más simples capaces de ejecutarlas o que constituyan instrucciones de computadoras disponibles.

2.3.2. Diseño descendente *(top-down)*

El **diseño descendente** (*top-down*) es el proceso mediante el cual un problema se descompone 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 relacionasen unas con otras mediante entradas y salidas de información. Es decir, se descompone el problema en etapas o estructuras jerárquicas, de forma que se puede considerar cada estructura desde dos puntos de vista: *¿qué hace*? y *¿cómo lo hace*?

Si se considera un nivel *n* de refinamiento, las estructuras se consideran de la siguiente manera:



Nivel *n:* desde el exterior
«¿qué hace?»

Nivel *n + 1:* Vista desde el interior
«¿cómo lo hace?»

El diseño descendente se puede ver en la Figura 2.7.

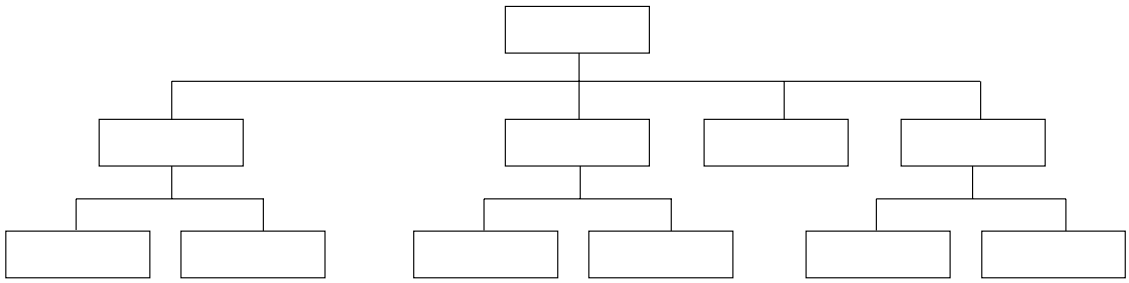


Figura 2.7. Diseño descendente.

2.3.3. Estructuras de control

Las *estructuras de control* de un lenguaje de programación son métodos de especificar el orden en que las instrucciones de un algoritmo se ejecutarán. El orden de ejecución de las sentencias (lenguaje) o instrucciones determina el *flujo de control*. Estas estructuras de control son, por consiguiente, fundamentales en los lenguajes de programación y en los diseños de algoritmos, especialmente los pseudocódigos.

Las tres estructuras de control básico son:

- *secuencia,*
- *selección,*
- *repetición.*

y se estudian en los Capítulos 5 y 6.

La programación estructurada hace los programas más fáciles de escribir, verificar, leer y mantener; utiliza un número limitado de estructuras de control que minimizan la complejidad de los problemas.

2.3.4. Teorema de la programación estructurada: estructuras básicas

En mayo de 1966, Böhm y Jacopini demostraron que *un programa propio* puede ser escrito utilizando solamente tres tipos de estructuras de control.

- *secuenciales,*
- *selectivas,*
- *repetitivas.*

Un programa se define como **propio** si cumple las siguientes características:

- *Posee un solo punto de entrada y uno de salida o fin para control del 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 (sin fin).*

La programación estructurada significa que

- El programa completo tiene un diseño modular.
- Los módulos se diseñan con metodología descendente (puede hacerse también ascendente).
- Cada módulo se codifica utilizando las tres estructuras de control básicas: secuenciales, selectivas y repetitivas (ausencia total de sentencias **GOTO**).
- *Estructuración y modularidad* son conceptos complementarios (se solapan).

2.4. CONCEPTO Y CARACTERÍSTICAS DE ALGORITMOS

El objetivo fundamental de este texto es enseñar a resolver problemas mediante una computadora. El programador de computadora es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático. A lo largo de todo este libro nos referiremos a la *metodología necesaria para resolver problemas mediante programas*, concepto que se denomina **metodología de la programación**. El eje central de esta metodología es el concepto, ya tratado, de algoritmo.

Un algoritmo es un método para resolver un problema. Aunque la popularización del término ha llegado con el advenimiento de la era informática, **algoritmo** proviene —como se comentó anteriormente— de *Mohammed al KhoWârizmi*, matemático persa que vivió durante el siglo IX y alcanzó gran reputación por el enunciado de las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales; la traducción al latín del apellido en la palabra *algorismus* derivó posteriormente en algoritmo. Euclides, el gran matemático griego (del siglo IV a.C.) que inventó un método para encontrar el máximo común divisor de dos números, se considera con Al Khowârizmi el otro gran padre de la algoritmia (ciencia que trata de los algoritmos).

El profesor Niklaus Wirth —inventor de Pascal, Modula-2 y Oberon— tituló uno de sus más famosos libros, *Algoritmos + Estructuras de datos = Programas*, significándonos que sólo se puede llegar a realizar un buen programa con el diseño de un algoritmo y una correcta estructura de datos. Esta ecuación será una de las hipótesis fundamentales consideradas en esta obra.

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto.