DISEÑO DESCENDENTE

En los Capítulos 1 y 2 hemos discutido el proceso de la programación y algunas estrategias de resolución de problemas. En esta sección juntaremos estas ideas para formar una metodología de desarrollo de programas. Recordemos que el proceso de la programación consta de una fase de resolución del problema y una fase de implementación.

Se le ha advertido para que no trate de saltarse la fase de resolución del problema, aunque para los problemas pequeños parezca que no hace falta y asegure usted que podría escribir directamente el programa. Existen buenas razones para insistir en realizar el proceso completo. Los problemas pequeños también tienen un gran número de detalles que deben cuidarse y una aproximación metódica es la mejor forma de asegurar que nada se olvida.

La práctica de aplicar la siguiente metodología a problemas pequeños, le preparará para aplicarlos a problemas mayores de programación, los cuales no se pueden disolver directamente. Como cualquier producto, su programa será legible, comprensible, fácilmente depurable y fácilmente modificable. Nosotros estructuramos nuestro método de programar a través de una técnica o método bien organizado conocido como *diseño descendente* (top-down). También llamada *refinamientos sucesivos* y *programación modular,* esta técnica nos permite usar el método "divide y vencerás". Recordemos que con este método un problema se divide en subproblemas que sean manejables. Después de resolver todos los subproblemas tendremos una solución del problema global.

Lo que hacemos en el diseño descendente es ir de lo abstracto (nuestra descripción o especificación del problema) a lo particular (nuestro código Pascal real).

Si la descripción del problema es una descripción en palabra vagamente establecida, entonces el primer paso es crear una *descripción funcional del problema.* Es decir, una descripción que establezca claramente lo que el programa ha de hacer. En muchos casos, esto significa un diálogo entre la persona que tiene el problema y el programador.

Módulos

Comenzaremos subdividiendo el problema en un conjunto de subproblemas. Luego, cada subproblema se divide a su vez en subproblemas. Este proceso continúa hasta que no se puede dividir más cada subproblema. Estamos creando una estructura jerárquica, también conocida como una estructura en árbol, de problemas y subproblemas llamados módulos funcionales. Los módulos de un nivel pueden llamar a los módulos del nivel inferior. Estos módulos son los bloques básicos de nuestros programas.

Los procedimientos y funciones predefinidas, tales como Write, Read y Abs, son ejemplos de módulos que resuelven sus problemas. Los subproblemas que resuelven se encuentran tan frecuentemente en programación que estos procedimientos y funciones han sido incorporados permanentemente en el lenguaje de

programación Pascal. La codificación de módulos de nuestro diseño como subprogramas es una forma de construir un programa en Pascal.

Una vez que un problema ha sido dividido en subproblemas, módulos o segmentos, podemos resolver cada módulo independientemente de los otros. Por ejemplo: un módulo podría leer los datos, otro podría imprimir los valores después del procesamiento. Varios módulos de procesamiento pueden llevar un total acumulativo, llevar la cuenta de los datos, detectar condiciones de error o hacer cálculos.

Nuestro árbol de diseño contiene sucesivos niveles de refinamiento (ver Figura 3-5). En la cabeza, o nivel 0, está nuestra descripción funcional del problema, los niveles inferiores son nuestros sucesivos refinamientos.

¿Cómo dividir el problema en módulos? Bien, pensemos por un momento cómo los humanos atacan normalmente un gran problema. Pasamos algún tiempo pensando en el problema globalmente, luego apuntamos los pasos principales. A continuación examinamos cada uno de los pasos principales, completando los detalles. Si no sabemos cómo realizar una determinada tarea, pasamos a la siguiente, teniendo en cuenta que hemos de volver a ella y anotando la que hemos saltado para cuando más adelante tengamos más información.

Así es como se debe enfocar un problema de programación. Pensar en cómo se resolvería el problema a mano. Escribir los pasos principales. Este es entonces el módulo principal. Comenzar a desarrollar los detalles de los pasos principales en los módulos del nivel 1. Si no se sabe cómo hacer algo, o se siente abrumado por los detalles, se le da un nombre a la tarea y se sigue adelante. Ese nombre puede expandirse más adelante como un módulo más inferior.

Este proceso continúa en tantos niveles como sean necesarios para expandir cada tarea hasta los más pequeños detalles. Esto puede llamarse la técnica de

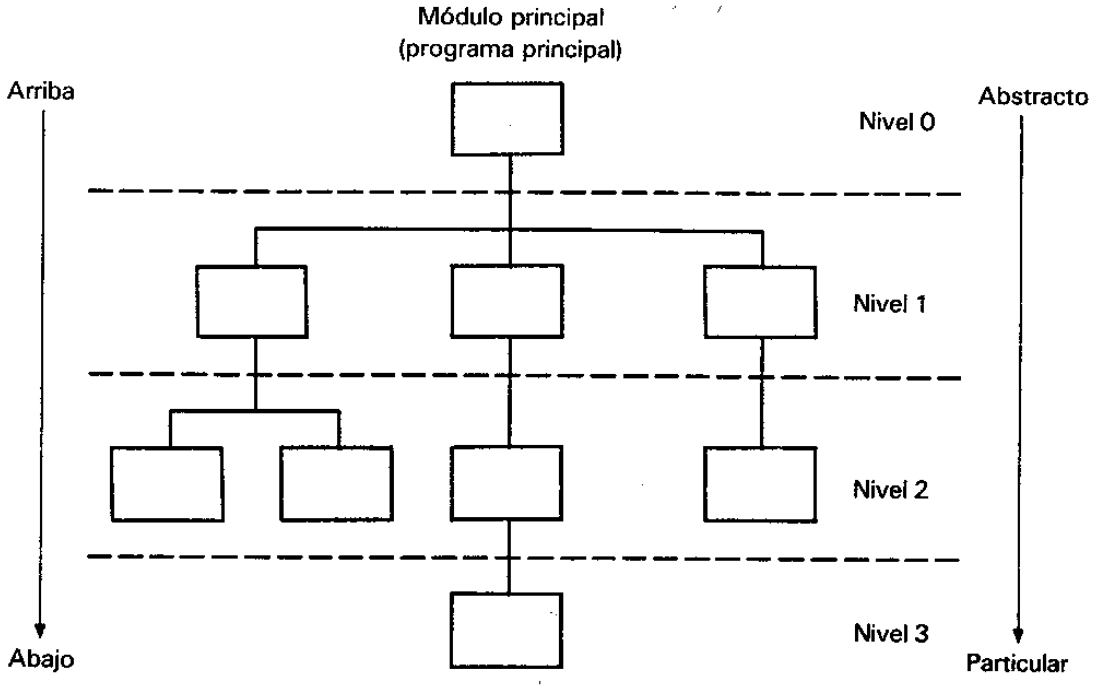


Figura 3-5. Jerarquía o estructura en árbol.

Scarlet O'Hara. Si una tarea es incómoda o difícil, aplazar el problema a un nivel inferior, no pensar en ello hoy, pensarlo mañana. De hecho nuestros mañanas han de venir, pero entonces este proceso global puede aplicarse a las subtareas algo difíciles. Eventualmente, el problema global se subdivide en unidades manejables.

Puede ayudarle el suponer que tiene un "amigo inteligente" el cual puede siempre resolver las partes de un problema que le presentan alguna dificultad. Simplemente dé un nombre a estas partes y déjelas al lado para que las resuelva su amigo inteligente. Una vez que haya resuelto todas las demás partes que sabe cómo hacer, puede volver atrás y examinar algunas de las cosas que ha dejado apartadas. Se sorprenderá agradablemente al ver cómo es mucho más sencilla una tarea cuando se concentra enteramente sobre ella. Lo que anteriormente le presentaba problemas dejará de hacerlo y usted se convertirá en su propio amigo inteligente. Si al resolver este subproblema se encuentra de nuevo frente a otra tarea poco abordable, déjela de nuevo para que su amigo inteligente la examine más adelante.

Apliquemos este proceso a la agradable tarea de planificar una gran fiesta. Una pequeña reflexión revelará que hay dos tareas principales: invitar a la gente y la preparación de la comida.

Un método de invitar a la gente podría ser coger la agenda de teléfonos y empezar a llamar a los amigos. Sin embargo, podría pronto confundirse acerca de quién había llamado, que teléfono estaba ocupado y qué ha dicho cada uno. Un método mucho mejor podría ser hacer una lista con las personas que se quiere invitar, ponerse la lista al lado y revisarla cada día para ver si se ha olvidado de algún buen amigo.

Luego, con la lista en la mano, se puede ir escribiendo el número de teléfono de cada uno. Ahora ya se puede empezar a llamar y a marcar las respuestas. Puede que lleve un tiempo llamar a cada uno, pero siempre se sabrá en dónde se está. Una vez que se tenga una estimación del número de personas que van a asistir, se podrá empezar a preparar la comida.

¡El cielo nos ayude si inmediatamente empezamos a cocinar! Sin una planificación previa este trabajo prodría ser angustioso. En vez de ello, dividamos esta tarea de planificar el menú y en preparar la comida.

Podemos ahorrarnos mucho tiempo y esfuerzo en esta tarea si nos aprovechamos de lo que otros han hecho y miramos los menús sugeridos en los libros de cocina. (En programación podríamos buscar en la literatura para ver si existen ya algoritmos que resuelvan este subproblema.) Una vez elegido un menú, podemos aplazar hasta más adelante el examen cuidadoso de las recetas. La hora de hacer esto es cuando estemos preparando la lista de compras. (Aplazamiento de los detalles hasta más adelante.)

El diagrama en árbol de la Figura 3-6 muestra el proceso de subdivisión hacia abajo (no se muestran todos los módulos del nivel 2).

Observe que un módulo de un nivel desarrolla una sentencia (tarea) del nivel anterior. Como humanos podríamos probablemente coger los módulos del nivel 2 y hacerlos a partir de esta descripción. Para un programa de computadora deberíamos subdividirlos en detalles mucho más finos. Por ejemplo, "escribir los nombres" podría hacerse con el siguiente nivel de detalle:

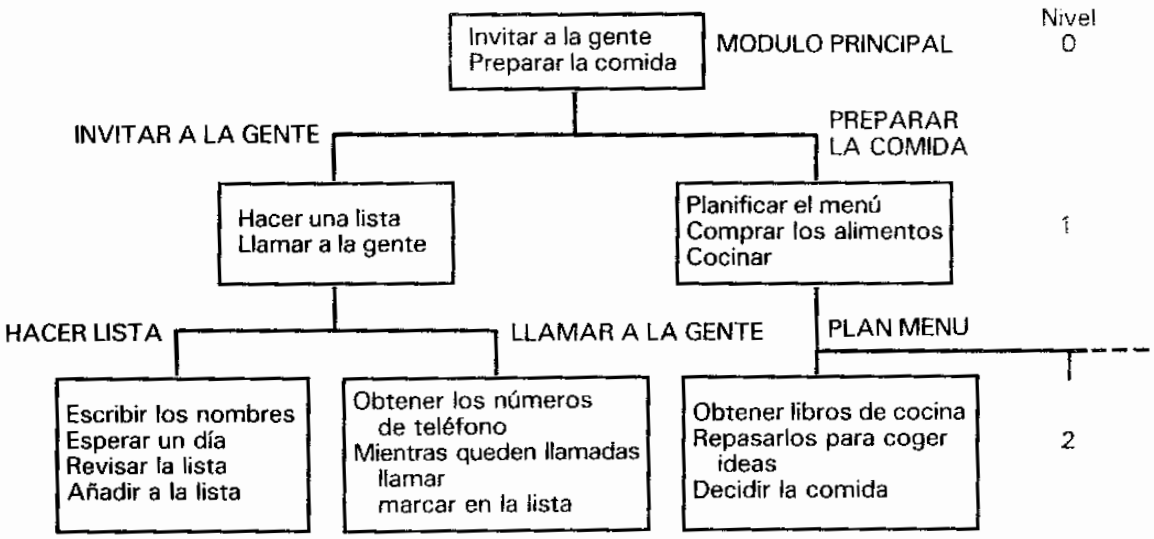


Figura 3-6. Planificación de una fiesta.

¿Tiene papel?
    No, obtenga papel.
¿Tiene una pluma?
    No, obtenga una pluma.
Coja la pluma.
Ponga la pluma en el papel.
Etcétera.

El diseño descendente para una fiesta puede ser muy distinto. Quizá si tiene una buena tienda especializada apuntada en el bloque, podría pedirle a ellos que organizaran la fiesta.

El módulo principal podría ser:

```
Invitar a la gente
Llamar a la tienda especializada
```

Dos diseños descendentes no serán exactamente iguales. No hay una única forma de escribir un diseño. Su diseño reflejará su propio estilo personal. Sin embargo, un "buen diseño" será modular, con las tareas agrupadas en unidades funcionales.

Dejemos ya la analogía social y examinemos de nuevo el proceso aplicado al problema de la programación de computadoras. Recuerde, el dominio es nuevo pero el proceso a seguir es algo que ya ha hecho antes.

El módulo principal especifica los nombres de las tareas. Cada nombre de una tarea ha de expandirse en un nivel inferior, a menos que tenga una correspondencia uno a uno con una sentencia del Pascal. Esto es verdad en cada nivel. Habrá tantos módulos principales en el nivel 1 como nombres de tareas en el nivel 0, y así sucesivamente para cada nivel.

La idea es aplazar los detalles. Poner el código real al nivel más bajo posible. Mientras menos se preocupe de la implementación real más puede concentrarse en las divisiones funcionales y en los algoritmos. Cuando se recorre el árbol del diseño se toman una serie de decisiones de diseño. Si una decisión es difícil o incorrecta (¡y lo será muchas veces!), es fácil volver atrás (ir por el árbol del diseño a un módulo de un nivel superior) y recapitular sobre lo que se necesita. A menos que tenga que desechar todo el diseño sólo tendrá que considerar esa pequeña parte. Puede tener que dar muchos paso intermedios y probar soluciones antes de llegar al diseño final.

El grueso del tiempo debe gastarse en el análisis y diseño de la solución. La codificación le llevará sólo una pequeña parte del tiempo. Si después de probar cada módulo con datos se asegura que cada uno funciona, entonces el programa debe funcionar cuando se pongan todos los módulos juntos. Un pequeño esfuerzo gastado en la comprobación de cada módulo puede ahorrar mucho esfuerzo en la depuración del programa.

Escribir un diseño descendente es similar a escribir un perfil para un artículo en castellano. Puede utilizar sentencias o pseudocódigo (una mezcla de castellano y estructuras de control en Pascal) en castellano para describir cada tarea o subtarea. Las estructuras de control del pseudocódigo son similares a las sentencias de selección e iteración mencionadas en los Capítulos 4 y 5. Cuando se introduzcan verá ejemplos de diseños descendentes que las utilizan. Para mostrar un pseudo-código, he aquí un módulo principal para un problema muy sencillo, el de encontrar la media de una lista de números.

```
Inicializar Suma y Contador a cero
WHILE haya más datos DO
  Obtener datos (leer datos)
  Añadir datos a Suma
  Incrementar Contador (sumar uno a Contador)
Poner Media a Suma/Contador
Imprimir Media
```

La estructura de control iterativa es la WHILE-DO y las tareas

```
Obtener datos
Sumar datos a Suma
Incrementar Contador
```

se hacen una y otra vez hasta que no haya más datos (números en la lista). Aunque todavía no hemos estudiado los bucles, el pseudocódigo para este algoritmo es muy claro y comprensible. Esta es una de las características de un algoritmo que esté bien escrito en pseudocódigo: el algoritmo es lo suficientemente claro para que

uno no tenga que ser un programador experimentado para comprender qué es lo que hace.

Debido a que éste es un problema muy sencillo, podemos codificar este módulo (cada sentencia puede traducirse directamente en una sentencia del Pascal). Este ejemplo muestra lo que es un pseudocódigo pero no muestra realmente la belleza del diseño descendente: la reducción de la complejidad de un problema mediante la división del problema en subproblemas (módulos) que puedan ser más fácilmente manejables.

¿Cuáles son las ventajas de un programa producido a partir de un diseño descendente? Es más fácil de comprender porque puede ser estudiado en piezas organizadas funcional y lógicamente, es más fácil de probar y de modificar.

Un programa debe reflejar el diseño descendente. Cualquier cambio en el diseño debe ser fácil de hacer en el programa. Cualquiera que lea el programa debe poder ver la descomposición del problema y la estructura de la solución.

Aunque la escritura de nuevos subprogramas es una forma de codificar módulos en nuestro diseño, no es la única forma de codificarlos. Se puede utilizar la técnica del diseño modular aunque no se conozca nada sobre los subprogramas. De hecho, el diseño modular es una técnica completamente independiente del lenguaje de programación. Aunque esté obligado a escribir un programa en FORTRAN, BASIC, COBOL o Ada, podrá usar estas técnicas. De hecho, como vimos con el ejemplo de la planificación de una fiesta, se puede utilizar esta técnica para resolver problemas en áreas distintas de la programación.

Metodología

El método del diseño descendente puede dividirse como sigue:

1. Analizar el problema.
2. Escribir el módulo principal.
3. Escribir los restantes módulos.
4. Reordenar y revisar lo que sea necesario.

Para implementar una solución de un problema en la computadora, se debe comenzar analizándolo y, si es necesario, reescribiendo la declaración del problema. El análisis del problema incluye el listado de los datos de entrada, la salida deseada y cualquier suposición que sea necesario hacer. Se pueden entonces listar los principales pasos de la solución informática. La lista de los pasos principales se convierte en el módulo principal del programa. Cada paso principal puede entonces ser examinado más detalladamente y posiblemente expandido para formar un módulo entero con su propio conjunto de pasos principales. Este proceso de expansión se llama *refinamiento.* Conforme se refina una solución, se va descubriendo que se necesita definir algunas variables que se vayan necesitando para que se pueda hacer su declaración cuando llegue la hora de codificar el programa. Examinemos más detalladamente cada uno de los pasos principales de la metodología basada en el diseño descendente.

1. Analizar el problema. Primero hay que comprender el problema, reescri birlo más claramente y completamente si es necesario. Comprender lo que se d (Entrada) y lo que se pide (Salida). Especificar los formatos de Entrada y Salida Listar las suposiciones que se hagan (si las hay). Pensar. ¿Cómo podría resolver e problema a mano? ¿Está familiarizado con algún problema parecido que haya resuelto antes? ¿Puede pensar en una buena analogía? Desarrollar algún algoritmo o plan general de ataque. Mantenga una visión global; no se detenga en los detalle de nivel inferior hasta que esté preparado para tratar con ellos.

2. Escribir el módulo principal. Se puede utilizar pseudocódigo o el castella no para establecer el problema en el módulo principal. Utilizar nombres de módu los para dividir el problema en áreas funcionales. Si el módulo principal tiene más de 10 ó 15 sentencias se está en un nivel demasiado bajo de detalle. Introducir en este punto cualquier estructura de control que se necesite (tal como bucle o selec ción). Si se necesita, revisar la lógica. Posponer los detalles a niveles inferiores siempre puede cambiar el módulo principal en futuros refinamientos.

No se preocupe si no sabe cómo resolver un módulo que no esté escrito en este momento. Simplemente suponga que su "amigo inteligente" conoce la respuesta y posponga su consideración hasta que se hagan más refinamientos. Todo lo que tiene que hacer en el módulo principal es dar los nombres de los módulos del nivel inferior que suministren ciertas funciones (resuelvan ciertas tareas). Utilice nombres con significado para los módulos.

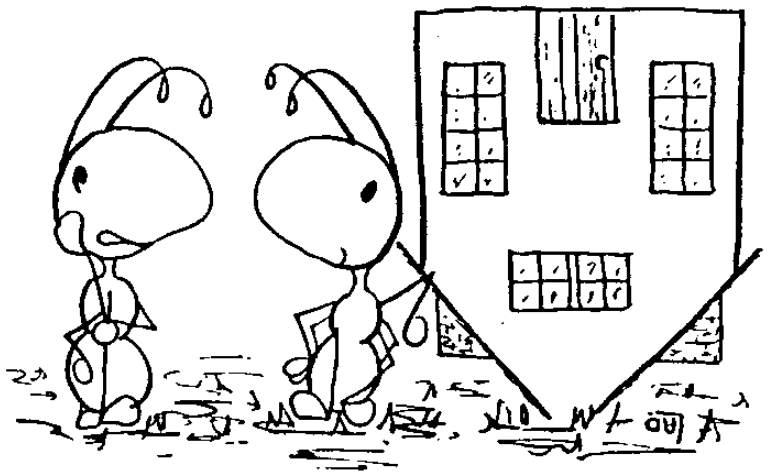
3. Escribir los restantes módulos. No hay un número fijo de niveles. Los módulos de un nivel pueden especificar más módulos del nivel inferior. Cada módulo debe estar completo, aunque puede referenciar a módulos que estén sin escribir. Haga sucesivos refinamientos añadiendo módulos de nivel inferior hasta que cada sentencia pueda traducirse directamente en una sentencia Pascal.

4. Reordenar y revisar lo que sea necesario. Planifique un cambio. No se preocupe de comenzar de nuevo. Pueden ser necesarios varios intentos y refinamientos. Trate de mantener la claridad. Exprésese simple y directamente.

Utilizaremos el siguiente esquema para nuestros diseños descendentes. El problema ejemplo que se da en la siguiente sección muestra la utilización de este esquema de un diseño descendente.

Establecimiento del problema.
Discusión.
Descripción de la entrada.
Descripción de la salida.
Suposiciones (si las hay).
Módulo principal.
Restantes módulos por niveles.

"¿Fue esto el resultado de tu primer intento de un diseño DESCENDENTE?"



Dibujo de M. LAD. TOPOLSKY

Documentación

Conforme se crea un diseño descendente, hay que desarrollar documentación para el programa. La documentación consiste en la escritura de descripciones, especificaciones, desarrollo y código real de un programa.

Una buena documentación ayuda en la lectura y comprensión y es inestimable cuando se modifica (mantenimiento) un programa. Si no se ha mirado un programa durante seis meses y se necesita volver a él y hacer algunos cambios, será muy importante que esté bien documentado. Por supuesto, ¡si cualquiera que no ha escrito el programa tiene que usar y modificar un programa, la documentación es indispensable!

La documentación externa del programa incluye las especificaciones, historia del desarrollo y diseño descendente. La documentación interna incluye formateado del programa, comentarios y código auto-documentado. Puede usar el pseudocódigo del diseño descendente como comentarios en el programa.

La documentación de la que hemos hablado es buena para el lector o mantenedor de sus programas. Sin embargo, si se usa un programa en un entorno de producción, debe darse también un manual del usuario.

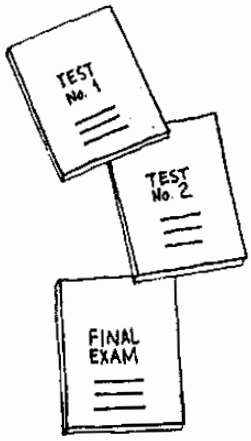
La documentación debe guardarse por fechas. Cualquier cambio que se haya hecho en un programa debe indicarse en toda la documentación pertinente.

La utilización de código auto-documentado hará sus programas más legibles.

Código auto-documentado. Un programa que contiene identificadores con significado, así como comentarios clarificadores usados juiciosamente.

Idealmente, un programa en Pascal debe ser legible aún por un no programador. El Apéndice G, Estilo de Programas, trata sobre algunos criterios de documentación de programas. Veamos ahora un par de problemas como ejemplos que mues-

tran la metodología de diseño descendente. Recuerde que es importante recorrer todos los pasos de la metodología de forma que no se subestime nada en el desarrollo de un diseño.



Resolución de problemas en acción

Problema: Encontrar la media ponderada de tres puntuaciones de exámenes. Los datos estarán en la forma de una puntuación entera de un examen seguida de su peso asociado, con cada par en una línea separada.

Discusión: Es frecuente dar pesos diferentes a los exámenes si se utiliza frecuentemente para obtener la nota del estudiante en un curso. Por ejemplo, si hay dos exámenes con un valor del 30 % cada uno y un examen final con un valor del 40 %, tomaríamos la nota del primer examen y la multiplicaríamos por 0,30, tomaríamos la nota del segundo examen y la multiplicaríamos por 0,30 y tomaríamos la nota final y la multiplicaríamos por 0,40. Luego sumaríamos estos tres valores para obtener la media ponderada.

Usaremos este algoritmo "manual" para resolver este problema.

Entrada: Tres grupos de datos cada uno compuesto de:

puntuación del examen (entero) peso (real)

Salida: Imprimir los datos de entrada con cabeceras (impresión del eco). Imprimir la media ponderada con explicación.

Suposiciones: Los tres pesos suman 1,00 y los datos de entrada son correctos (no hay error de comprobación)

MODULO PRINCIPAL Nivel 0

```
Obtener datos
Imprimir datos
Encontrar media
Imprimir media
```

OBTENER DATOS Nivel 1

```
Escribir 'Introducir primera nota y peso de examen.
Leer Test1, Peso1
Escribir 'Introducir segunda nota y peso de examen.
Leer Test2, Peso2
Escribir 'Introducir tercera nota y peso de examen.
Leer Test3, Peso3
```

RPA

IMPRIMIR DATOS

```
Imprimir cabecera
Escribir Test1, Peso1
Escribir Test2, Peso2
Escribir Test3, Peso3
```

ENCONTRAR MEDIA

```
Med = Test1 * Peso1 + Test2 * Peso 2 + Test3 *
  Peso3
```

IMPRIMIR MEDIA

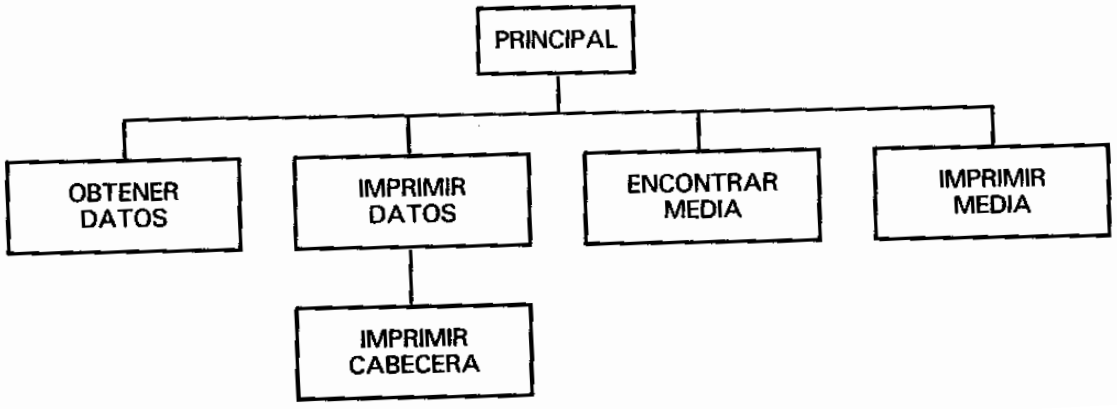
```
Escribir 'Media ponderada = ', Med
```

IMPRIMIR CABECERA

Nivel 2

```
Escribir 'Nota test   Peso'
```

Diagrama de estructuras de módulos:



RPA

Lista de variables del algoritmo:

Nombre	Tipo de datos	Uso
Test1	Integer	Puntuación para el primer test
Test2	Integer	Puntuación para el segundo test
Test3	Integer	Puntuación para el tercer test
Peso1	Real	Peso del primer test
Peso2	Real	Peso del segundo test
Peso3	Real	Peso del tercer test
Med	Real	Media ponderada de los tests

```
PROGRAM MediaTest (Input, Output);

(* Programa para encontrar la media ponderada de tres puntuaciones
   de test *)

VAR
  Test1,          (* Puntuación para el primer test  *)
  Test2,          (* Puntuación para el segundo test *)
  Test3:          (* Puntuación para el tercer test  *)
    Integer;
  Peso1,          (* Peso del primer test            *)
  Peso2,          (* Peso del segundo test           *)
  Peso3,          (* Peso del tercer test            *)
  Med:            (* Media ponderada de los tests    *)
    Real;

BEGIN (* MediaTest *)
  (* Obtener datos *)
  Writeln('Introducir primera nota y peso de examen.');
  Readln(Test1, Peso1);
  Writeln('Introducir segunda nota y peso de examen.');
  Readln(Test2, Peso2);
  Writeln('Introducir tercera nota y peso de examen.');
  Readln(Test3, Peso3);
  Writeln;
  (* Imprime cabecera *)
  Writeln(' Nota Test   Peso ');
  Writeln;
  (* Imprime datos *)
  Writeln(Test1:8, Peso1:12:2);
  Writeln(Test2:8, Peso2:12:2);
  Writeln(Test3:8, Peso3:12:2);
  Writeln;
  (* Calcula la media *)
  Med := Test1 * Peso1 + Test2 * Peso2 + Test3 * Peso3;
  (* Imprime la media *)
  Writeln('Media ponderada = ', Med:7:2)
END.  (* MediaTests *)
```

RPA

El programa está diseñado para una entrada de datos interactiva desde un terminal (observe el uso de los mensajes de petición). Si cuando el programa se ejecutó el usuario introdujo los datos de entrada

```
90  0.30
85  0.25
78  0.45
```

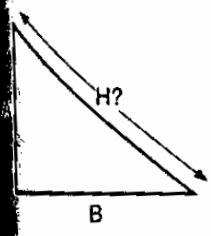
el diálogo con el usuario podría ser algo como lo que sigue

```
Introducir primera nota y peso de examen.
90  0.30
Introducir segunda nota y peso de examen.
85  0.25
Introducir tercera nota y peso de examen.
78  0.45

 Nota Test     Peso

    90         0.30
    85         0.25
    78         0.45

Media ponderada =     83.35
```



Resolución de problemas en acción

Problema: Determinar la longitud de la hipotenusa de un triángulo rectángulo dadas las longitudes de los otros dos lados. Los datos se darán en forma de dos números reales, uno para cada una de las longitudes de los lados.

Discusión: Probablemente conocerá el teorema de Pitágoras, el cual establece que el cuadrado de la hipotenusa de un triángulo rectángulo es igual a la suma de los cuadrados de los otros dos lados. Por tanto, para calcular la longitud de la hipotenusa, debe obtener el cuadrado de las longitudes de los lados, sumar los cuadrados y luego calcular la raíz cuadrada de la suma. Así es como resolvería el problema con una calculadora. Usaremos la misma técnica para nuestra solución por computadora.

Entrada: Dos números reales, uno para cada lado del triángulo distinto de la hipotenusa.

RPA

Salida:

Imprime los datos de entrada con un mensaje que identifica cada número (impresión del eco).
Imprime la longitud de la hipotenusa con un mensaje de identificación.

Suposiciones: Las dos longitudes son números reales positivos (no hay comprobación de error).

MODULO PRINCIPAL Nivel 0

```
Obtener datos
Imprimir datos
Encontrar longitud de la hipotenusa
Imprimir longitud de la hipotenusa
```

OBTENER DATOS Nivel 1

```
Escribir 'Introducir las longitudes de las dos caras.'
Leer LongDeA, LongDeB
```

IMPRIMIR DATOS

```
Escribir 'La longitud de cara la A es', LongDeA
Escribir 'La longitud de cara la B  es', LongDeB
```

ENCONTRAR LONGITUD DE LA HIPOTENUSA

```
Calcular SumaDeCuadrados
Hipotenusa = Sqrt(SumaDeCuadrados)
```

IMPRIMIR LONGITUD DE LA HIPOTENUSA

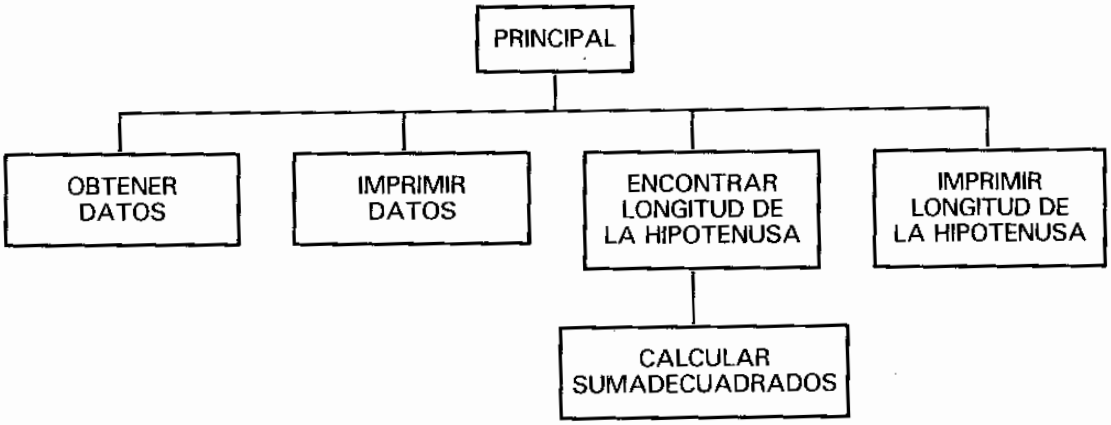
```
Escribir 'La longitud de la hipotenusa es', Hipotenusa
```

RPA

CALCULAR SUMADECUADRADOS Nivel 2

SumaDeCuadrados = Sqr(LongDeA) + Sqr(LongDeB)

Diagrama de estructuras de módulos:



Lista de variables del algoritmo:

Nombre	*Tipo de datos*	*Uso*
LongDeA	Real	Longitud de uno de los lados
LongDeB	Real	Longitud del otro lado
Calcular SumaDeCuadrados	Real	Suma de los cuadrados de los dos lados
Hipotenusa	Real	Longitud de la hipotenusa

```
PROGRAM Triángulo (Input, Output);

(* Este programa calcula la longitud de la hipotenusa de un
   triángulo rectángulo, dadas las longitudes de los otros
   dos lados *)

VAR
  LongDeA,                  (* Longitud de uno de los lados          *)
  LongDeB,                  (* Longitud del otro lado                *)
  Calcular SumaDeCuadrados, (* Suma de los cuadrados de los dos lados*)
  Hipotenusa:               (* Longitud de la hipotenusa             *)
    Real;
```

RPA

```
BEGIN (* Triángulo *)
  (* Obtiene los datos *)
  Writeln('Introducir las longitudes de las dos caras.');
  Readln(LongDeA, LongDeB);
  (* Imprime los datos *)
  Writeln;
  Writeln('La longitud de la cara A es ', LongDeA:7:4);
  Writeln('La longitud de la cara B es ', LongDeB:7:4);
  (* Calcula la suma de los cuadrados *)
  Calcular SumaDeCuadrados := Sqr(LongDeA) + Sqr(LongDeB);
  (* Calcula la longitud de la hipotenusa *)
  Hipotenusa := Sqrt(Calcular SumaDeCuadrados);
  (* Imprime la longitud de la hipotenusa *)
  Writeln('La longitud de la hipotenusa es ', Hipotenusa:8:4)
END.  (* Triángulo *)
```

Los datos de este programa es la entrada: Va separada del programa lee interactivamente cuando se ejecuta el programa. Si cuando se ejecu programa el usuario introdujo los datos de entrada

```
95.019  42.91
```

el diálogo con el usuario podría ser el siguiente:

```
Introducir las longitudes de las dos caras.
95.019  42.91

La longitud de la cara A es 95.0190
La longitud de la cara B es 42.9100
La longitud de la hipotenusa es 104.2587
```