Capítulo 5

DISEÑO MODULAR: SUBPROGRAMAS

1. INTRODUCCIÓN

En la construcción de los ejercicios se han estado utilizando subprogramas desde que se introdujeron las instrucciones de entrada y salida, *escribir* y *leer* en pseudocódigo, y *print, read, sqrt* en Java. Además, en la realización del pseudocódigo generalizado y como método de trabajo, se ha aplicado la técnica de división del problema en módulos más simples, aunque estos han debido especificarse, sentencia a sentencia, cuando se ha escrito el pseudocódigo detallado. En este capítulo se aprenderá a construir subprogramas, funciones y procedimientos, que permitirán simplificar la construcción de programas largos y complejos.

Por otro lado, en capítulos anteriores se ha hablado de las tres estructuras básicas de control de flujo del programa: secuencial, de bifurcación y repetitiva. Ahora se estudiará la última de las estructuras de control, los subprogramas.

2. DISEÑO MODULAR: ENFOQUE TOP-DOWN

El diseño modular es una metodología de desarrollo de programas, que surge de la necesidad de abordar un problema complejo dividiéndolo en otros más sencillos, que se pueden llamar subproblemas, consiguiendo así comprender y abordar mejor el problema completo. De esta forma, después de resolver todos los subproblemas se obtiene la solución del problema global.

Este método o filosofía de diseño se implementa a través de una técnica llamada **Top-Down**, conocida también como **diseño descendente** o **refinamiento por pasos sucesivos**, por tratar el problema desde lo abstracto, desde arriba (Top), es decir, desde la descripción o especificación del problema, a lo particular, o abajo (Down), es decir, a cada línea de código.

Esta no es la única técnica aplicable al diseño modular. Por ejemplo, también existen técnicas que van de lo particular hacia arriba, es decir, que construyen programas complejos a partir de componentes particulares. Esta técnica se conoce como **Botton-Up** y será usada en programación orientada a objetos.

3. MÓDULOS. DIAGRAMAS JERÁRQUICOS

La técnica de descomposición consiste en comenzar dividiendo el problema en un conjunto de subproblemas, a continuación se dividen estos en otros más sencillos, y así sucesivamente. El proceso seguirá hasta que no se pueda o no convenga dividir más cada trozo, porque ya resulta suficientemente sencillo el desarrollo de estos. De esta forma, se crea una estructura jerárquica de problemas y subproblemas, llamados **módulos funcionales**. La representación gráfica de esta estructura se conoce como **Diagrama de Estructura de Módulos** *(***DEM***)*.

La estructura jerárquica en árbol que resulta contiene distintos niveles de refinamiento, tal como se observa en la siguiente figura.

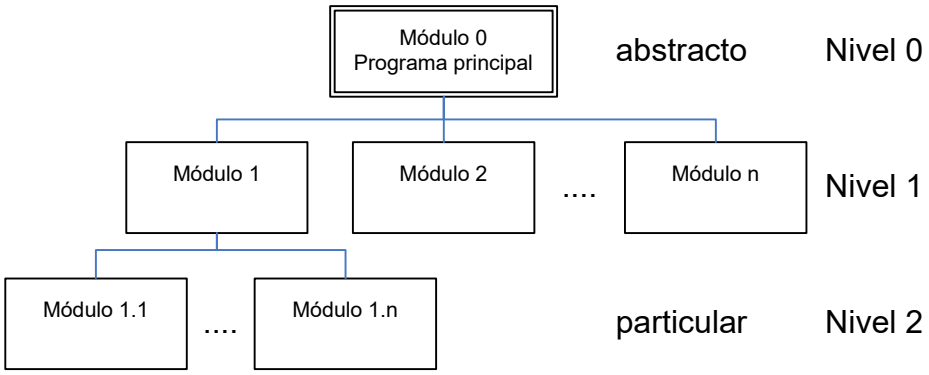

Figura 5.1. Diagrama de Estructura de Módulos (DEM)

La descomposición modular contribuye a las características deseables para la programación estructurada, presentando además algunas ventajas respecto a la programación sin módulos. Por ejemplo, va a permitir resolver cada módulo de forma independiente y, si no se sabe resolver alguno de ellos, puede dársele nombre, pasar al diseño de otro módulo y, más adelante, cuando se tenga más información sobre el primero, se retoma nuevamente para darle solución. De esta forma, no se detiene el trabajo de diseño por el desconocimiento de algún proceso en particular. Esta técnica se conoce como **estrategia de Escarlata O'Hara**. [DALE y WEEMS 1991]

En resumen, entre las ventajas que supone el uso de programación modular se encuentran:

- Facilita que la resolución del problema la lleven a cabo varios programadores, gracias a la división del problema en distintos módulos.

- Contribuye a la comprensión y eleva el grado de legibilidad de los algoritmos.
- Reduce el tiempo y coste del desarrollo de software al permitir la reutilización de módulos anteriormente desarrollados en proyectos nuevos.
- Agiliza la depuración de programas al permitir que cada módulo pueda depurarse de manera independiente.
- Por último, favorece el mantenimiento y modificación de los módulos ya diseñados y, por consecuencia, de los programas.

Así como no hay una única solución para un problema, no hay una única forma de diseñar un modelo descendente. El estilo personal de cada programador quedará reflejado en el diseño. Sin embargo, cabe decir que todo buen diseño modular tendrá las tareas agrupadas en unidades funcionales.

¿Cómo dividir el problema en módulos? Se afronta la división de la misma forma en que se afronta la resolución de grandes problemas en la vida cotidiana. La idea principal es aplazar los detalles de diseño, dejando el código real detallado, es decir, las sentencias directamente ejecutables para implementarlo en el nivel más bajo posible. Esto permitirá la concentración del diseñador en las divisiones funcionales y en los algoritmos generales. El analista y diseñador debe pasar la mayor parte del tiempo de trabajo analizando y diseñando, lo que se transformará en poco tiempo codificando y corrigiendo errores.

Resumiendo, entre las pautas a seguir para aplicar la filosofía descendente se encuentran:

- Pasar un tiempo pensando en el problema globalmente.
- Escribir los pasos principales, especificando tanto la estructura principal del programa como los módulos que realiza.
- Examinar cada uno de los pasos anteriores completando los detalles.
- Si no se sabe realizar una determinada tarea, se le da un nombre y se pasa a la realización de la tarea siguiente, teniendo en cuenta que habrá que ocuparse de ella más adelante para resolverla.
- Este proceso continúa en tantos niveles como sea necesario, hasta completar el diseño.

No todos los módulos de un DEM se convertirán en subprogramas. Cabe representar otro gráfico de la división del problema en subprogramas llamada **Diagrama de Descomposición Funcional** (**DDF**). Aunque hay varios tipos de DDF, el más utilizado representa las funciones del sistema en distintos niveles de abstracción, pero no los flujos de datos entre ellos.

¿Cuáles de los módulos se convierten en subprogramas?. Debe haber un compromiso entre los niveles de modularidad y la complejidad de los problemas que

deben resolverse. Una vez diseñado el Diagrama de Estructuras de Módulos cualquier módulo, en general, puede ser resuelto como un subprograma. La decisión de convertirlo deberá basarse en si será más fácil o no, comprender el programa global como resultado de dicha decisión, si se prevé que pueda ser utilizado en la resolución de otros problemas, o varias veces en el mismo problema, y si la conversión no supone un esfuerzo informático añadido inútil.

Por ejemplo, no sería rentable, en cuanto a la construcción de subprogramas se entiende, que un módulo **Elegir y validar Opción,** como el del ejercicio *MenuOperaciones* del capítulo 3, se convirtiera en subprograma, puesto que existen directamente órdenes en Java que resuelven la tarea y el paso a subprograma supondría un coste adicional como, establecer las variables de enlace, más requerimiento de memoria, incremento del tiempo de ejecución, etc. Es lo que se denomina **coste de interfaz**. Por otro lado, tampoco es un módulo que se pudiera utilizar en otros ejercicios puesto que es una validación de una entrada particular de un ejercicio en concreto.

Puede observarse el ejemplo comentado, desarrollado como módulo y como subprograma:

a) El módulo se diseñará con un bucle *Repetir...Mientras*, más adecuado para validaciones que el bucle *Mientras*, y su inclusión en el programa principal sería lineal, es decir, bastará con incluir las órdenes que lo componen directamente en el lugar que ocupa el módulo en el programa, tal como se ha venido haciendo hasta ahora.

```
Elegir y validar Opcion
Repetir
   leer (opcion) /*Opcion es una variable del programa
                  principal ya declarada.*/
   opcion = Mayuscula (opcion)
Mientras (opcion != 'S' y opcion != 'R' y opcion != 'M' y
           opcion != 'D' y opcion != 'F')
Fin EyVO
```

b) Se puede convertir el módulo anterior en subprograma diseñándolo como función o como procedimiento. Se va a diseñar como función y su inclusión en el programa principal se hará en forma de llamada a dicha función.

   Aquí aparecen conceptos nuevos que precisamente van a tratarse en este capítulo.

```
Carácter ElegirValidarOpcion(  ) /*Sin variables de
                                      enlace*/
Inicio
Entorno: /*Opcion es una variable de este subprograma, por
          tanto aquí habrá que declararla.*/
   Carácter opcion
```

```
Repetir
   leer (opcion)
   opcion = Mayuscula (opcion)
Mientras (opcion != 'S' y opcion != 'R' y opcion != 'M' y
          opcion != 'D' y opcion != 'F')
Devolver (opcion)
Fin EyVO
```

4. METODOLOGÍA. DOCUMENTACIÓN

4.1. Metodología

Tal como se comentó en el primer capítulo, en la fase de resolución del problema deberán tratarse los siguientes puntos:

1) **Análisis**. Se realiza el análisis, señalando el proceso que se ejecuta, las entradas y las salidas del programa principal, las suposiciones si las hay y el estudio de los bucles más conflictivos.

2) **Solución General**. Se implementa el algoritmo que representa el módulo principal en pseudocódigo generalizado, en el que aparecerán las llamadas a los módulos que contenga y la estructura general del programa. Se diseñarán tantos niveles de refinamiento de este algoritmo como se estime conveniente, relegando los detalles del diseño a niveles inferiores. Recuérdese que en cualquier momento puede convenir modificar el módulo principal al plantearse sucesivos refinamientos.

   Se dará nombre de módulos a todas las tareas, incluso en el caso en que no se sepan resolver. Esto no deberá suponer una preocupación en primera instancia, más adelante, se retomará para diseñarlo. Puede ser que se encuentre algún módulo similar diseñado con anterioridad o incluso, que otra persona lo resuelva y ya estará solucionada la cuestión.

   Tampoco se deberá olvidar que el código de los programas deberá ser autodocumentado por lo que los módulos deberán tener nombres alusivos a lo que realizan.

   No hay límites en el número de niveles de refinamiento de un problema. Se añadirán módulos de nivel inferior mientras resulte rentable el esfuerzo que conlleva, desde el punto de vista informático.

3) Escribir el código correspondiente a cada uno de los módulos que contenga el programa. Si se realiza por niveles, resultará más fácil la lectura y el seguimiento del algoritmo completo, es decir, se escribirá el código de los módulos de un nivel y, si en ellos vuelven a aparecer nuevos módulos, se describirá el código de estos y, así sucesivamente, hasta que se de por terminado el nivel de detalle.

4) Se realiza el *Diagrama de Estructura de Módulos*. Esto forma parte de la documentación externa del programa y facilitará la comprensión del problema global.

5) Se señalan aquellos que se vayan a convertir en subprogramas. De todos estos, se estudiará la interfaz correspondiente.

6) Se hace el pseudocódigo detallado del algoritmo, donde aparecerá el código correspondiente a los módulos no convertidos en subprogramas y las llamadas a los que si se han convertido.

7) Se realiza el *Diagrama de Descomposición Funcional*. Este también forma parte de la documentación del programa.

8) Por último, se ejecuta la traza, efectuando la revisión y corrección de lo que sea necesario.

En cualquier fase, puede que se vuelva repetidamente a cualquiera de los puntos anteriores para modificar o reestructurar, pudiendo incluso retornar al punto 1 si se observan errores importantes de diseño.

A partir de ahora, el estudio de los bucles se realizará con menos profundidad que en capítulos anteriores, puesto que se supone ya adquiridas las destrezas suficientes como para saber construir un bucle de forma más dinámica. Sin embargo, se hará especial hincapié en el estudio de la estructura modular que forman los subprogramas.

4.2. Documentación

La **documentación** del software es un tema muy importante que no está siempre tratado e incorporado adecuadamente al análisis y al diseño de aplicaciones, al que añade, entre otras ventajas, legibilidad y comprensión. Por esta razón, se hará especial hincapié a lo largo del libro en su uso, recordando especialmente que no se programa para uno mismo, sino para los demás.

A la documentación utilizada hasta ahora en la creación de los programas, se añadirán descripciones y especificaciones de cada módulo, y otra documentación necesaria en el proceso de construcción de software.

1. **Documentación externa** que contiene:
   - **Especificaciones**: ¿qué hace cada módulo? ¿en qué contexto?.

     Para los módulos que vayan a convertirse en subprogramas, deberá detallarse:
     - El proceso que realiza.
     - Las precondiciones.
     - Las entradas.

- Las salidas.
- Las postcondiciones.

Además de externamente como ayuda para el uso de subprogramas, es habitual incluir esta información internamente, junto al código fuente cuando se realiza la definición de los subprogramas. De esta forma servirá de documentación a los demás diseñadores.

- **Desarrollo**. Es la historia de las modificaciones sufridas por el programa, si las hubiera.

- **Esquema gráfico** del diseño descendente si lo hubiera (DEM y/o DDF).

2. **Documentación interna** que contiene código autodocumentado con:
   - Comentarios en la llamada a subprogramas desde el programa principal o desde cualquier subprograma.
   - Comentarios en los subprogramas incluyendo:
     - Etiqueta de cada subprograma con su nombre.
     - Breve descripción de lo que realiza.
     - Comentarios para explicar el propósito de parámetros formales y variables locales.

EJERCICIO PRÁCTICO DE APLICACIÓN

Realizar el pseudocódigo de un programa que mediante un menú de opciones nos permita realizar algunas operaciones con números positivos menores que 100, tales como calcular el factorial, tabla de multiplicar hasta 10 y si el número es o no múltiplo de 3. El usuario podrá repetir cuantas veces desee una determinada opción, además de repetir el proceso completo.

Análisis. En el programa principal se presentará un menú de opciones donde el usuario podrá elegir una de ellas incluida la de salida. El proceso general será iterativo hasta que el usuario desee abandonarlo. Aunque la VCB sea un centinela y por tanto, habría que hacer una lectura anticipada que permita no realizar ningún proceso si elegimos esta opción, usaremos un bucle con condición al final, controlado por el dato centinela que será la opción para salir del programa.

Por otro lado, deberá tenerse en cuenta que cada opción deberá poder repetirse, sin salir de ella, las veces que desee el usuario. Por tanto, se necesitará otro bucle similar al anterior controlado por el centinela ‘N’, que será lo que deba contestar el usuario para salir.

Las condiciones de entrada a los bucles serán detalladas en los sucesivos niveles de refinamiento.

Entradas. La opción de menú, el número para realizar los cálculos y la respuesta a seguir realizando una determinada operación matemática dentro de una opción.

Salidas. Los resultados de las distintas operaciones en pantalla: cálculo del factorial, tabla de multiplicar, si el número es o no múltiplo de 3 y los distintos mensajes para completar y reforzar el código.

Suposiciones: La opción de menú y los números se consideran enteros y la posibilidad de seguir de tipo carácter, por tanto, no habrá comprobación de error de los tipos de datos, tan sólo habrá que comprobar los rangos permitidos. Quiere esto decir, que corresponderá al usuario del programa la responsabilidad de introducir el tipo de datos adecuado a cada entrada.

Primer nivel de refinamiento

//Pseudocódigo generalizado de nivel 0

Programa principal generalizado

Nivel 0

```
   //Nombre del Programa: MenuFMM3
Inicio
 Repetir
    Mostrar MENU Elegir y Validar OPCION
    Si(opcion<> 4)
      Según (opcion)
         Para opcion == 1
            Calcular Factorial
         Para opcion == 2
            Visualizar Tabla Multiplicar
         Para opcion == 3
            Comprobar  si es Múltiplo de 3
      Fin_según
    finsi
 Mientras (OPCION != 4)
Fin PP
```

En el siguiente nivel de refinamiento, se detallarán cada uno de los módulos de los que se compone el programa principal anterior.

MÓDULOS

Nivel 1

⇒ **Mostrar MENU Elegir y Validar OPCION**

```
  Inicio
    Repetir
      PresentarMENU
      Leer (opcion)
    Mientras (opcion < 1 O opcion > 4)
  Fin MM
```

/* No hay inconveniente en poner *mientras opcion incorrecta, pero en algún momento habrá que estudiar esta condición.*/

⇒ **Calcular Factorial**

```
  Inicio
    Repetir
```

```
            Leer y Validar SEGUIR
            Si (seguir== 's')
                Obtener y Validar NUMERO
                Hacer FACTORIAL
                Mostrar Resultado Factorial
            finsi
          Mientras (SEGUIR == 'S')
        Fin CF                  // No hay inconveniente en poner mientras desee seguir
```

```
⇒ Visualizar Tabla Multiplicar
    Inicio
      Repetir
        Leer y Validar SEGUIR
        Si (seguir== 's')
            Obtener y Validar NUMERO
            Calcular y Mostrar Tabla
        finsi
      Mientras (SEGUIR == 'S')
    Fin VTM
```

```
⇒ Comprobar  si es Múltiplo de 3
    Inicio
      Repetir
        Leer y Validar SEGUIR
        Si (seguir== 's')
            Obtener y Validar NUMERO
            Hacer MULTIPLO3
            Mostrar Resultado Múltiplo
        finsi
      Mientras (SEGUIR == 'S')
    Fin CM3
```

Como puede observarse, al describir los algoritmos generales de los módulos de primer nivel, han aparecido nuevos módulos y nuevas estructuras de control de flujo que tendrán que ser detalladas, refinando nuevamente el problema. Algunas de estas estructuras y módulos forman parte de la estructura principal del programa, por lo que deberán aparecer en el pseudocódigo generalizado refinado en primer nivel.

//Pseudocódigo generalizado de nivel 1

Programa principal generalizado

```
Inicio
 Repetir
   //Mostrar MENU Elegir y Validar OPCION
        Repetir
          Presentar MENU
          Leer (opcion)
```

```
        Mientras (opcion < 1 O opcion > 4)
    //Fin MM
    si (opcion <>4)
      Según (opcion)
        Para opcion == 1
            //Calcular Factorial
            Repetir
              Leer y Validar SEGUIR
              Si (seguir ==’s’)
                Obtener y Validar NUMERO
                Hacer FACTORIAL
                Mostrar Resultado Factorial
              finsi
            Mientras (SEGUIR == ’S’)
            //Fin CF
        Para OPCION == 2
            //Visualizar Tabla Multiplicar
            Repetir
              Leer y Validar SEGUIR
              Si (seguir ==’s’)
                Obtener y Validar NUMERO
                Calcular y Mostrar Tabla
              finsi
            Mientras (SEGUIR == ‘S’)
            //Fin VTM
        Para OPCION == 3
            //Comprobar  si es Múltiplo de 3
            Repetir
              Leer y Validar SEGUIR
              Si (seguir ==’s’)
                Obtener y Validar NUMERO
                Hacer MULTIPLO3
                Mostrar Resultado Múltiplo
              finsi
            Mientras (SEGUIR == ‘S’)
            //Fin CM
      Fin_según
    finsi
  Mientras (opcion != 4)
Fin PP
```

En el siguiente nivel de refinamiento se detallará cada uno de los nuevos módulos aparecidos en el pseudocódigo generalizado escrito anteriormente.

Nivel 2

```
⇒ Presentar MENU
    Inicio
      Escribir (“1- Calcular Factorial
              2 –Visualizar tabla de multiplicar
              3 – Comprobar si es múltiplo de 3
```

```
                4 – Salir del programa")
    Fin PM
```

⇒ **Obtener y Validar NUMERO**

```
    Inicio
      Repetir
        Escribir ("Introduzca un número positivo")
        Leer (numero)
      Mientras (numero < 0  O  numero > 100)
    Fin OVN
```

⇒ **Leer y Validar SEGUIR**

```
    Inicio
      Repetir
          Escribir ("Desea seguir (S/N)")
          Leer (seguir)
          Pasar A Mayúsculas SEGUIR*
      Mientras (seguir != 'S'  Y seguir != 'N')
    Fin LVS
```

***** *Este módulo sirve para que sea transparente al usuario el pulsar mayúsculas o minúsculas al elegir opción. Cuando se opte por un lenguaje de programación, en nuestro caso Java, se estudiará como se resuelve este módulo, si lo tiene incorporado el lenguaje o si debe ser implementado por el diseñador.*

⇒ **FACTORIAL**

```
    Inicio
      Para(contador  =  numero;  mientras  contador>1;
           contador = contador – 1)
          factorial = factorial * contador
      FinPara
    Fin F
```

⇒ **Mostrar Resultado Factorial**

```
    Inicio
      Escribir("El  factorial  de  ---  es  ---  ",numero,
               factorial)
    Fin MRF
```

⇒ **Calcular y Mostrar Tabla**

```
    Inicio
      Para  (contador  =  1;  mientras  contador  <=  10;
            contador = contador + 1)
          resultado = numero * contador
          Escribir ("--- * --- = ---", numero, contador,
                    resultado)
      FinPara
    Fin CMT
```

⇒ **MULTIPLO3**

```
    Inicio
      resto = numero % 3 /*Almacena el resto de dividir
```

```
                                    el número entre 3.*/
        Si (resto == 0)
            multiplo = 'S'
        En otro caso
            multiplo = 'N'
        FinSi
    Fin M3
```

⇒ **Mostrar Resultado Múltiplo**

```
    Inicio
      Si (multiplo  == 'S')
          Escribir ("El número ---   es múltiplo de 3",
                    numerO)
      En otro caso
          Escribir ("El número --- no es múltiplo de 3",
                    numero)
      Finsi
    Fin MRM
```

Como puede observar, en este segundo nivel no aparecen nuevos módulos, por tanto, se puede pasar a escribir el diagrama de estructuras de módulos del ejercicio y el pseudocódigo detallado a continuación.

Diagrama de Estructura de Módulos

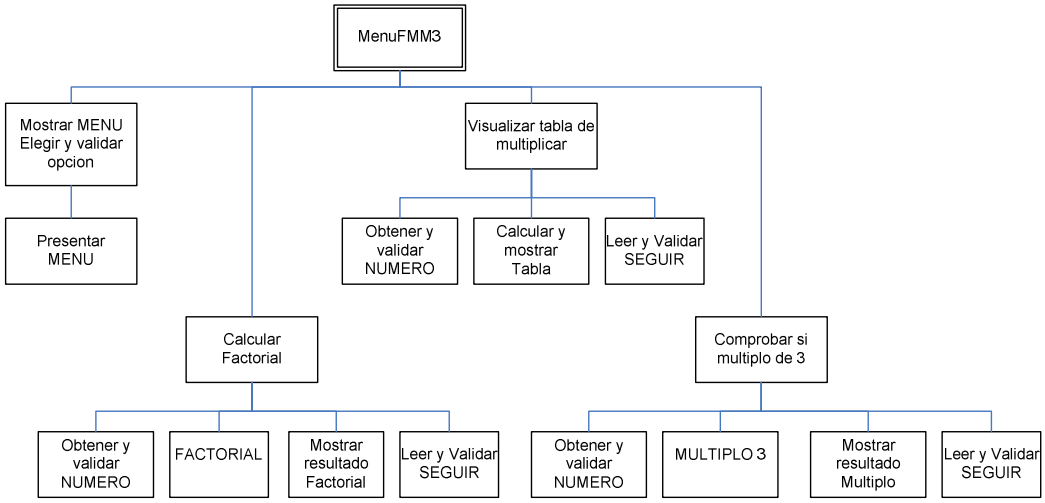

Figura 5.2. Diagrama de Estructura de Módulos de MenuFMM3

Una vez detallados todos los módulos, y con el DEM delante, se pensará cuáles de ellos son idóneos para ser transformados en subprogramas, sean funciones o procedimientos, según se establezca en el diseño de su interfaz. Parece oportuno que

estos módulos sean los siguientes: **Presentar MENU, FACTORIAL, Calcular y Mostrar Tabla** y **MULTIPLO3**.

Pseudocódigo

```
ENTORNO:
    /*Nombre del programa: MenuFMM3*/
    VARIABLES:
     Carácter: opcion, seguir
     Entero: numero, factorial, múltiplo, resultado
Programa principal detallado
 Inicio
  Repetir
    /*Mostrar MENU Elegir y Validar OPCION*/
    Repetir
       Presentar MENU
       Leer (opcion)
    Mientras (opcion < 1 o opcion > 4)
    Si (opcion<>4)
      Según (opcion)
      Para opcion == 1
            /*Calcular Factorial*/
        Repetir
          /*Leer y Validar SEGUIR*/
          Repetir
            Escribir ("Desea seguir (S/N)")
            Leer (seguir)
            Pasar A Mayúsculas seguir
          Mientras (seguir != 'S'  Y seguir != 'N')
          Si (seguir=='s')
           /*Obtener y Validar NUMERO*/
           Repetir
              Escribir ("Introduzca un número positivo")
              Leer (numero)
           Mientras (numero < 0  o  numero > 100)
           Hacer FACTORIAL
           /*Mostrar Resultado Factorial*/
           Escribir("El factorial de --- es --- ",numero,
                    factorial)
          finsi
        Mientras (seguir == 'S')
      //fin opción 1
      Para opcion == 2
            /*Visualizar Tabla Multiplicar*/
        Repetir
          /*Leer y Validar SEGUIR*/
          Repetir
            Escribir ("Desea seguir (S/N)")
```

```
                Leer (seguir)
                Pasar A Mayúsculas SEGUIR
              Mientras (seguir != 'S'  Y seguir != 'N')
              Si (seguir=='s')
                /*Obtener y Validar NUMERO*/
                Repetir
                 Escribir ("Introduzca un número positivo")
                 Leer (numero)
                Mientras (numero < 0  o  numero > 100)
                Calcular y Mostrar Tabla
              finsi
            Mientras (seguir == 'S')
            //fin opción 2
          Para opcion == 3
               /*Comprobar  si es Múltiplo de 3*/
            Repetir
              /*Leer y Validar SEGUIR*/
              Repetir
                Escribir ("Desea seguir (S/N)")
                Leer (seguir)
                Pasar A Mayúsculas SEGUIR
              Mientras (seguir != 'S'  Y seguir != 'N')
              Si (seguir=='s')
                /*Obtener y Validar NUMERO*/
                Repetir
                  Escribir ("Introduzca un número positivo")
                  Leer (numero)
                Mientras (numero < 0  o  numero > 100)
                Hacer MULTIPLO3
                /*Mostrar Resultado Múltiplo*/
                Si (multiplo = = 'S')
                 Escribir ("El número ---  es múltiplo de 3",
                           numero)
                En otro caso
                 Escribir ("El número --- no es múltiplo de 3",
                           numero)
                Finsi
              Finsi
            Mientras (seguir == 'S')
            //fin opciòn Múltiplos
          Fin_según
        FINSI
      Mientras (opcion != 4)
    Fin PP
```

Se aconseja pasar la traza y realizar la depuración del ejercicio. La traza deberá ejecutarse sobre cada uno de los módulos por separado y sobre el programa principal de los distintos niveles. Recuerde que el objetivo es detectar errores lo más

tempranamente posible y subsanarlos, por lo que es bueno hacerlo en todas las fases antes de codificar el algoritmo.

Puede observarse que se han dejado sin sustituir, en el desarrollo del pseudocódigo detallado del ejercicio, los módulos que aparecen en negrita, precisamente los que se van a desarrollar como subprogramas: **`Presentar MENU`**, **`FACTORIAL`**, **`Calcular y Mostrar Tabla`** y **`MULTIPLO3`**.

Diagrama de Descomposición Funcional (DDF)

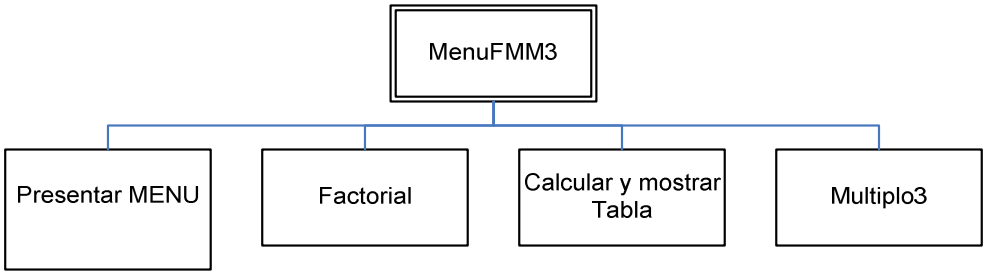

Figura 5.3. Diagrama de Descomposición Funcional de MenuFMM3

5. PROGRAMA PRINCIPAL Y SUBPROGRAMAS

5.1. El programa principal

Tiene las siguientes características:

- Describe la semántica de la solución completa del problema.
- Consta principalmente de estructuras de control básicas y de llamadas a subprogramas. Las llamadas a subprogramas son indicaciones al procesador para que pase el control de flujo al subprograma y vuelva de nuevo al programa principal cuando acabe el código de dicho subprograma.
- El programa principal contiene, además de sentencias de control directamente ejecutables por el procesador (leer, escribir, sentencias simples, si, para, mientras, según, etc.), declaraciones de objetos, definiciones de tipos o cualquier otra sentencia.

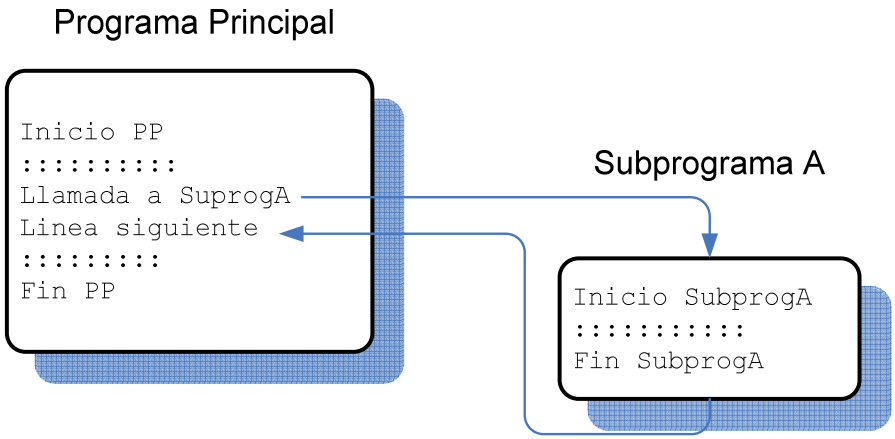

Figura 5.4. Flujo de control en llamadas a subprogramas

Cuando en el programa principal el compilador encuentre la *Llamada a SubprogA*, pasará la ejecución al *Inicio SuprogA*, realizará todo el código correspondiente a este subprograma y, cuando encuentre *Fin SubprogA*, volverá el flujo de control al programa principal en la línea siguiente a la llamada al subprograma (Fig. 5.4).

5.2. Subprogramas

Tienen las siguientes características:

- Un subprograma es una estructura de control de flujo.
- La llamada a un subprograma es una instrucción que transfiere el control al subprograma. En el siguiente capítulo, se verá que en Java esta instrucción está formada por el nombre del subprograma seguido de una lista de parámetros.
- La estructura de un subprograma es la de un programa principal sencillo, con algunas diferencias en el encabezamiento y en la finalización. Cualquier subprograma también podrá contener llamadas a otro u otros subprogramas.
- Un subprograma se ejecuta cuando es llamado por el programa principal o por otro subprograma.
- Un subprograma puede ser llamado cuantas veces sea necesario, tanto por el programa principal como por otros subprogramas. Esto significa que puede estar dentro de un bucle o aparecer en diferentes puntos del programa que lo llama.

5.3. Cohesión y Acoplamiento

En una aplicación deben decidirse de antemano cuales son los requisitos de calidad que deben cumplirse y el orden de importancia entre ellos, en cuyo aseguramiento y evaluación se utilizan distintas métricas. Para evaluar el diseño de la estructura modular resultante en un programa debe realizarse la medida de calidad mediante la *cohesión* y el *acoplamiento*. El objetivo consiste en reducir al máximo el acoplamiento y aumentar la cohesión de los módulos.

COHESIÓN

Entendiendo por ente cualquier componente de un subprograma: instrucciones, datos, llamadas a otros subprogramas, etc., la cohesión estudia la relación existente entre los entes de un mismo módulo. Se puede definir como la medida del grado de identificación de un módulo con una funcionalidad concreta.

La cohesión es una característica importante en la construcción de subprogramas, ya que influye no sólo en la calidad, sino también en el coste del producto resultante. Existe una escala de cohesión que determina el mayor o menor grado de calidad de los subprogramas. En esta escala un subprograma altamente coherente debe realizar una única tarea.

ACOPLAMIENTO

El acoplamiento mide el grado de interrelación de los subprogramas que constituyen un programa. Para alcanzar un acoplamiento mínimo ningún módulo debe conocer los detalles internos del resto de subprogramas. Esto significa que los subprogramas deben ser absolutamente independientes, de manera que si un módulo necesita algo de otro deben comunicarse mediante los parámetros y pasarse exclusivamente lo que es indispensable.

Para conocer más sobre el tema véase [PIATTINI et al. 2003].

6. TIPOS DE SUBPROGRAMAS

6.1. Según su ubicación

Dependiendo del lugar donde esté definido el código de un subprograma, es decir, la situación que tiene respecto a la del programa o módulo que lo llama, se puede establecer la siguiente clasificación:

SUBPROGRAMAS INTERNOS

Tienen las siguientes características:

- Su código figura dentro o al mismo nivel que el programa principal. Pero siempre lo haremos al mismo nivel.

- Se declaran y/o definen físicamente antes del programa principal.

- Las llamadas se hacen por su nombre y dependen del lenguaje de compilación que se use para implementarlo.

Por ejemplo:

```
import java.io.*;
import java.lang.*;
public class Ejemplo
{

   //Subprograma diseñado como función
   public static int sumar(int n1,int n2)
   {
       ...//definición del código del subprograma
   }
   public static void main (String[] args)
   {
       ...
       sum = sumar(2,7); //Llamada al subprograma
       ...
   }
```

SUBPROGRAMAS EXTERNOS

Tienen las siguientes características:

- Figuran físicamente separados del programa principal, de manera que hay que incluir el lugar donde se encuentran. En Java este lugar se conoce como librería. Por ejemplo, `print` es un subprograma (Método) que se encuentra en la clase `System` que a su vez pertenece a la librería `java.lang`. Esta librería ("Packages", paquete en java) debe ser incluida o importada, esto es llevado a cabo mediante el calificativo *import* (de esto hablaremos en la Unidad siguiente).

- Pueden compilarse separadamente e, incluso, codificarse en un lenguaje distinto al del programa principal.

- Las llamadas, como los subprogramas internos, también se hacen por su nombre y dependiendo del lenguaje.

- Generalmente, no se dispone de su código fuente, pues el lenguaje no suele incorporarlo, sino que incorpora los códigos compilados. Si se trata de subprogramas externos desarrollados por el propio diseñador también termina usándose los códigos compilados de estos.

Las siguientes son llamadas a subprogramas con distintos lenguajes

COBOL `CALL nombre-subprograma`

Pascal	`EXECUTE (nombre-subprograma)`
C	`nombre-subprograma (parámetros);`
Java	`nombre-subprograma (parámetros);`

6.2. Según el valor de retorno

Según el valor que devuelven al programa, o subprograma, que lo llama cuando se ejecuta su código, los subprogramas pueden ser:

PROCEDIMIENTOS

No tienen valor asociado al nombre, o lo que es lo mismo, no tienen ningún valor de retorno al programa o subprograma que hizo la llamada. Por tanto, una llamada a un procedimiento no puede aparecer a la derecha ni a la izquierda de un símbolo de asignación.

FUNCIONES

Tienen valor asociado a su nombre, es decir, tienen valor de retorno al programa o subprograma que realizó la llamada. Por tanto, una llamada a función puede aparecer a la derecha, pero no puede aparecer a la izquierda de un símbolo de asignación.

En Java se conocen todos los subprogramas como funciones, aunque incorpora las herramientas necesarias para implementar procedimientos (Capítulo 6).

7. VARIABLES DE ENLACE O PARÁMETROS

Todo programa utiliza unos datos de entrada y produce unos resultados. Los datos de entrada proceden de las unidades de entrada y los resultados son enviados a las unidades de salida. En los subprogramas, los datos de entrada vienen del programa principal o del subprograma que llama y los resultados son enviados al programa principal o al subprograma que llama. Para este flujo de información se utilizan **variables de enlace** o **parámetros**.

7.1. Tipos de parámetros

PARÁMETROS FORMALES

Representan la forma en que programa principal y subprograma se comunican. Los parámetros formales son **variables** u otros **objetos de datos locales** de los subprogramas. Se utilizan para la recepción de los datos que llegan con la llamada desde el programa que la realiza o para la emisión de resultados hacia éste.

Aparecen declarados en la cabecera del subprograma entre los paréntesis que siguen al nombre y son fijos para cada subprograma.

Los identificadores usados en la lista de parámetros formales no pueden aparecer como identificadores de otros objetos locales en el subprograma.

En el siguiente ejemplo, *num* es una variable de enlace de *Raiz*, en este caso, parámetro formal y se comporta como variable local de este subprograma.

```
Raiz (num)
 Inicio
  .........
  .........
 FinR
```

En el siguiente ejemplo *base* y *exponente* son parámetros formales, que se comportan como variables locales del subprograma *Potencia.*

```
Potencia (base,exponente)
 Inicio
  ..........
  ..........
 FinP
```

PARÁMETROS ACTUALES O REALES.

Son **variables** o **constantes** u otros **objetos de datos locales**, enviados en cada llamada a un subprograma desde el programa o subprograma que lo llama.

Éstos parámetros aparecen listados (escritos) en la llamada a un subprograma.

Por ejemplo:

Programa principal que llama a un subprograma

```
PROGRAMA PRINCIPAL
  Inicio
  ...........
  Raiz (numero) //llamada a Raiz, le pasa Numero
   ..........
  FinPP
```

numero es el parámetro real de la llamada, variable de enlace del programa principal con *Raiz.*

Subprograma que llama a otro subprograma

```
MODULO_M (...)
 Inicio
  ..........
 Potencia (bas,3) //llama a Potencia y le pasa bas y
                  bas y 3 como parámetros reales.
  ..........
 FinMM
```

En este ejemplo, *numero***,** además de ser parámetro formal de la llamada, es una variable local del programa principal que llama a *Raiz***,** mientras que *bas* y *3* son objetos locales de *MODULO_M*. En este caso, los parámetros son una variable y una constante.

Entre parámetros reales y parámetros formales existe una correspondencia, en cuanto al tipo de datos, orden y, en principio también, en cuanto al número de ellos. El flujo de comunicación se realiza en orden, del primer parámetro real al primero formal, del segundo real al segundo formal, etc. Los tipos de datos de cada parámetro y el número de ellos deberán ser iguales en la llamada y en la cabecera del subprograma llamado, aunque existen lenguajes de programación, entre ellos C++ y Java, que permiten que el número de parámetros de la llamada pueda variar.

7.2. Paso de parámetros

Como se ha señalado anteriormente, el flujo de información entre programas y subprogramas, o entre unos subprogramas y otros, se realiza con el paso de parámetros. A través de estos, el programa principal puede pasar valores a los subprogramas para que los use en sus procesos y los subprogramas pueden generar, aunque no necesariamente, resultados como salida, es decir, pueden devolver valores al programa principal o al subprograma que los llama.

El paso de parámetros se realiza conceptualmente de dos formas diferentes: por **Valor** y por **Referencia**.

PASO POR VALOR

El paso de parámetros por valor **se utiliza para suministrar datos de entrada** a un subprograma, desde el programa principal o el subprograma que lo llama, de modo que el valor del parámetro real o actual se copie en el parámetro formal correspondiente.

El subprograma al que se pasa un parámetro por valor, trabaja con una copia de este, por lo tanto, el parámetro real no podrá ser modificado por el subprograma de ninguna manera. Si el subprograma contiene alguna sentencia que modifica el parámetro pasado por valor, se cambiaría tan sólo la copia, pero no el parámetro real de la llamada.

En definitiva, se puede derivar que el paso por valor evita que se produzcan cambios no intencionados en los parámetros actuales o reales.

Ejemplo:

> La profesora entrega una copia sobre un tema importante a los alumnos, que estudiarán con ella, pudiendo modificarla a su antojo, incluso añadiendo al margen las anotaciones que consideren oportunas. Imagine que el texto contiene ciertos errores y que los alumnos los corrigen en sus respectivas copias. Ninguno de ellos es consciente de las correcciones hechas por los demás compañeros, puesto que cada uno las ha realizado sobre su documento. Es más, el original que quedó en manos de la

profesora, no ha sufrido ninguna de las modificaciones realizadas en las distintas copias.

PASO POR REFERENCIA

Es llamado también por algunos autores **paso por variable** [DALE y WEEMS 1991]. **Se usa indistintamente para suministrar datos de entrada y recibir datos de salida**. El programa principal manda al subprograma la **referencia en memoria** del parámetro actual al correspondiente parámetro formal, con lo que el subprograma que utiliza la variable así pasada, lo hace como si fuera propiamente suya, por tanto, puede modificarla a conveniencia.

El paso por referencia se evitará en la medida de lo posible, puesto que puede producir efectos laterales, es decir, cambios indeseables en los parámetros reales.

Recuerde que el nombre de una variable es la identificación o referencia de la dirección de memoria donde se almacenará dicha variable.

Cuando un parámetro es pasado por referencia se pondrá delante de su declaración en la lista de parámetros y en el programa principal la palabra **referencia** o, simplemente, **ref**. Esta regla no es ningún estándar, sólo una forma de indicar en lenguaje natural que el paso se realizará de este modo. Es el lenguaje de programación elegido quién determinará la sintaxis correspondiente.

Ejemplo:

Siguiendo con el ejemplo anterior, suponga que la profesora entrega su original para que lo use un alumno que lo necesita y que se quedó sin copia en el reparto. La profesora indica al alumno el lugar (referencia) donde se encuentra dicho original, el cajón de su mesa, por ejemplo, y comunica al alumno que, cuando termine de usarlo, deberá dejarlo en el mismo sitio donde se encontraba. En este caso, cuando el alumno realice las correcciones oportunas sobre el documento, sí que quedará constancia sobre el original, puesto que sobre éste ha trabajado y la profesora tendrá su original corregido en el lugar en el que lo dejó inicialmente.

Pero, suponga que el alumno se dedica a escribir en el margen bromas, dibujos o cualquier otro tipo de texto de los que suelen encontrarse en los apuntes de algunos alumnos. En este caso, se producirá el efecto indeseado al que se hacía referencia anteriormente.

El siguiente cuadro resume los tipos de parámetros:

Tipo de parámetro	Utilización
Actual o Real	Se encuentra en una sentencia de llamada a un subprograma. Puede ser pasado por valor o por referencia a un parámetro formal.
Formal por valor	Se encuentra en la lista de parámetros en la cabecera de un subprograma y recibe una copia del valor almacenado en el parámetro actual correspondiente de la llamada al

	subprograma.
Formal por Referencia	Se encuentra en la lista de parámetros en la cabecera de un subprograma y recibe la referencia del parámetro actual correspondiente de la llamada al subprograma.

¿Por qué las dos formas de pasar parámetros? ¿Por qué no sólo por referencia? Porque cuando se desea que el parámetro sea modificado y que quede constancia de ello en el programa principal, el paso deberá realizarse por referencia y si, por el contrario, no se está interesado en las modificaciones, si el paso se hace por valor se evitarán manipulaciones indeseadas del parámetro real.

¿Es mejor pasar los parámetros por valor? Es lo aconsejable, puesto que el paso por valor previene los efectos laterales, y hace más claro el diseño y más fácil la lectura. Por tanto, se usará siempre que sea posible, dejando las referencias para el caso en que el contexto del problema a resolver así lo exija.

8. OBJETOS LOCALES Y GLOBALES

Conviene repasar algunos conceptos ya vistos anteriormente y particularizarlos para el uso de subprogramas. En el capítulo anterior se habló de *ámbito* como la parte de programa y/o subprograma en que son conocidos los objetos, y por tanto, pueden ser utilizados. De *tiempo de vida* como el tiempo durante el cual un objeto ocupa un lugar en memoria, pudiendo ser local o global.

Tiempo de vida global

Un objeto tiene **tiempo de vida global** cuando, una vez definido, su espacio de almacenamiento y sus valores permanecen en memoria mientras se ejecute el programa, y desaparecerá cuando termine la ejecución. Se dice que el objeto nace cuando se define, vive mientras se ejecuta y muere cuando acaba el programa.

Tiempo de vida local

Un objeto tiene **tiempo de vida local** si nace y se le asigna memoria al pasar el flujo de control del programa por el bloque de código en que el objeto está definido. Acabada la ejecución de dicho bloque, desaparecerá el espacio en memoria que tenía asignado, es decir, muere. El objeto sólo vive en el trozo de código donde se ha definido y no en el resto del programa.

Efectos laterales

Se define un **efecto lateral** como cualquier repercusión que ejerce un subprograma sobre otro que no forma parte de la interfaz definida específicamente entre ellos, es decir, es un efecto no deseado.

Objetos locales

Se declaran dentro del programa principal o dentro de cualquier subprograma que lo necesite y, por tanto, su uso está restringido a ellos. Su tiempo de vida es local,

existen mientras se ejecuta el código del programa o subprograma donde se han declarado.

En caso de que un subprograma necesite usar un objeto local de otro subprograma, hay que pasarlo como parámetro.

Objetos globales

Se declaran antes del comienzo del programa principal. Su ámbito se extiende a él mismo y a los subprogramas que contenga el mismo archivo. Tienen tiempo de vida global, su vida durará mientras dure la ejecución del programa.

Como estos objetos son conocidos por todos los módulos, pueden ser usados por todos sin necesidad de pasarlos como parámetros.

Teniendo en cuenta lo anterior, no se aconseja usar variables globales con la intención de evitar el paso de parámetros a subprogramas, a no ser estrictamente necesario, porque:

- Su uso en un subprograma restringe la utilización de éste, dificultando con ello la reutilización de código. Que no sea necesario pasar como parámetros las variables globales parece una ventaja, pero los subprogramas que las usen dependerán siempre de esas variables declaradas en el programa que los contiene, por lo que los subprogramas no podrán ser exportados para su utilización en otras aplicaciones, ni podrán ser incluidos en librerías para uso público.

- Sus posibles efectos laterales: El uso de variables globales es una mala práctica de programación que puede conducir a errores de difícil localización, que usualmente aparecen como efectos laterales indeseados.

A continuación, se realizará un ejemplo para ilustrar los efectos laterales producidos por el uso de variables globales:

Ejemplo

Realizar un algoritmo que lea caracteres de la entrada estándar y cuente la cantidad de líneas escritas, teniendo en cuenta que el final del proceso será cuando se introduzca un '.' (punto). Se considera que la línea acaba con '-' o cuando se han escrito 80 caracteres.

NOTA: El siguiente código no es en absoluto metodológicamente correcto, pero, se ha diseñado así para ilustrar el caso de los efectos laterales indeseados debidos al uso de objetos de datos globales en los programas. Por otro lado, como el ejemplo es muy simple y el objetivo observar los efectos laterales, se ha pasado directamente a especificar el algoritmo en pseudocódigo detallado.

Pseudocódigo Detallado

```
ENTORNO:
  /*Nombre del programa: Efectos_laterales*/
```

```
  Variables:
    carácter car, //global
    entera cont, //global

  Procedimientos
    ContarCarLinea () /*Cuenta caracteres por línea.*/
```

PROGRAMA PRINCIPAL

```
 Inicio
  cont=0
  Repetir
    Llamar a ContarCarLinea () /*La inicialización y
                               actualización de la VCB está
                               dentro del subprograma*/
    cont = cont+1
  Mientras (car <> '.')
  Escribir ("número de líneas", cont)
 Fin PP

 /*ContarCarLinea. Subprograma que cuenta caracteres por línea.
 Considera cada línea de 80 caracteres o acabadas en '-' o '.'.
 No tiene entradas, ni salidas, ni precondiciones, ni
 postcondiciones. */
```

ContarCarLinea ( ) *//No tiene parámetros formales*

```
 Inicio
  cont=0
  Repetir
    Escribir ("Escribe un carácter(para finalizar '.')")
    Leer(car)
    cont= cont+1
  Mientras (car <> '-'y car <> '.'y cont <= 80)
  Escribir("el número de caracteres/línea", cont)
 FinContar_Car
```

Para observar el efecto del uso de variables globales en el programa anterior, siga la traza del programa principal, compruebe que funciona. A continuación y sin tener en cuenta el programa principal, siga la traza del subprograma, igualmente observará que funciona. Entonces, ¿dónde está el error? Haga lo mismo pero siguiendo la traza paso a paso, es decir, siguiendo el flujo de control del programa como lo haría el compilador. Puede comprobar que cada vez que se ejecuta el programa, el número de líneas es igual al número de caracteres leídos en la última línea más 1.

Advierta que no ha sido necesario el paso de parámetros, puesto que al tratarse de variables globales son conocidas tanto por el programa principal como por el subprograma.

Por otro lado, el ejemplo también pone de manifiesto que el código del subprograma depende de las variables globales que ha usado, por tanto, no es exportable, no se puede usar en ningún otro programa, ni añadirlo a ninguna librería.

9. PROCEDIMIENTOS Y FUNCIONES

Existen dos tipos de subprogramas, llamados funciones y procedimientos, que tienen características bien diferenciadas.

PROCEDIMIENTOS

Son subprogramas internos o externos.

Sin valor asociado al nombre o, lo que es igual, sin valor de retorno. Que no tengan valor de retorno, no significa que no dispongan de datos de salida al módulo que los llama. Significa que, donde aparezca la llamada por su nombre, no existirá ningún valor cuando, después de ejecutarse su código, vuelva el flujo de control al programa.

Se declaran antes del programa principal. Si son procedimientos internos la declaración se hará en el mismo archivo del programa y, cuando son externos, su declaración o prototipo se encuentra en un archivo, que en C se llama **archivo de cabecera** y que deberá incluirse en el programa que use dicho procedimiento.

Su formato de declaración es el siguiente:

nombreProcedimiento (tipos_datos_parámetros)

nombreProcedimiento. Es el identificador que se usa para nombrar al procedimiento. Recuerde que debe ser autodocumentado.

tipos_datos_parámetros. Cada tipo de dato de los parámetros formales separados por comas. Aparecerán tantos tipos como parámetros formales haya.

La llamada se hace desde el programa principal, según el siguiente formato:

nombreProcedimiento (lista_parámetros_reales)

lista_parámetros_reales. Son los nombres de las variables del programa principal o subprograma que llama, que se usan como parámetros reales**.** Estarán separados por comas y habrá tantos como parámetros formales en la cabecera del subprograma.

¿Qué se diseña como procedimiento? Los módulos que se implementarán como procedimientos serán aquellos que no estén obligados a devolver un valor explícito asociado a su nombre al programa principal o al subprograma que lo llama.

FUNCIONES

Tienen las siguientes características:

- Son subprogramas internos o externos.

- Tienen un valor asociado al nombre. Es decir, devuelven un sólo valor en cada ejecución al módulo que realiza la llamada. Este valor sustituirá al nombre de la función en el módulo que llama. Ocurrirá cuando acabada la ejecución del subprograma se vuelva el control a dicho programa.

- Si son funciones internas se declararán antes del programa principal, pero si son externas, al igual que los procedimientos, su declaración estará en otro archivo que deberá incluirse en el programa que use dicha función.

La declaración tiene el siguiente formato:

tipo nombreFunción (tipos_datos_parámetros)

Donde *nombreFunción* es el identificador que referencia a la función, y *tipo* es el tipo del valor que devuelve la función y que puede ser cualquier tipo de datos básico o definido por el usuario.

tipos_datos_parámetros son los tipos de datos de los parámetros formales separados por comas.

La llamada se hace dentro del módulo que llama. El formato más habitual es:

nombreVariable = nombreFunción (Lista_parámetros_reales)

nombreVariable es una variable declarada en el módulo que llama y en la que se almacenará el valor devuelto por la función. Esta variable deberá ser de un tipo compatible con el valor devuelto por la función. Como la llamada a la función es un *rvalue*, puede utilizarse en cualquier expresión que lo admita.

Lista_parámetros_reales son los nombres de los parámetros reales separados por comas. Deberán ser tantos como parámetros formales.

¿Qué se diseña como función? Se diseñarán como funciones aquellos subprogramas que realicen cálculos y que produzcan unos resultados basados en ellos necesarios para el programa principal.

Es importante considerar que los subprogramas, ya sean funciones o procedimientos, no deberán leer ni escribir, salvo que ese sea su objetivo. Por ejemplo, si el subprograma es un menú, deberá escribir en pantalla puesto que ese es su único objetivo, pero si el subprograma resuelve el factorial de un número, no deberá leer el número, ni pintar el resultado, esto será responsabilidad del programa que lo llame.

10. DISEÑO DE LA INTERFAZ

La descripción de los parámetros de un subprograma, el tipo de dato que devuelve y el nombre del subprograma, deben estar especificados antes de que pueda ser usado en una llamada. Estos tres elementos constituyen la interfaz del subprograma [COHOON y DAVIDSON 2000], y establecen la vía comunicación entre el módulo

que realiza la llamada y el subprograma llamado. Para **diseñar la interfaz** deben considerarse los siguientes puntos:

1. Todos los objetos que necesita el subprograma y que tiene el módulo que hará la llamada, ya sea el principal u otro subprograma, se consideran **Necesidades** del subprograma.

2. Aquellos objetos que produce o calcula el subprograma, y que deben ser devueltos al módulo que realiza la llamada, se conocen como **Devoluciones** del subprograma.

3. Por último, aquellos objetos que tiene el módulo que llama y que necesita el subprograma, pero que este manipule, modifique y deba devolver a quien lo llame, son las **Necesidades/Devoluciones** del subprograma.

Por tanto, se puede resumir en el siguiente cuadro:

	Tipos de datos	**Cómo se pasan**
Necesidades	Datos de entrada	Parámetros pasados por valor. Excepto algunos objetos por propia naturaleza, por ejemplo, los arrays.
Devoluciones	Datos de salida	Parámetros pasados por referencia si el subprograma es un procedimiento, o *valor devuelto* asociado al nombre del subprograma si es una función.
Necesidades/Devoluciones	Datos de entrada y salida	Parámetros pasados por referencia.

Restricciones:

Después del estudio anterior se pensará si los parámetros de entrada, si los hubiera, deben cumplir algún requisito o restricción.

Las restricciones podrán convertirse o no en precondiciones. Pero en cualquier caso, de existir restricciones supondrá que se deberá generar el código correspondiente para la verificación de dichas restricciones. Esto será realizado, bien por el programa principal, caso de haber precondiciones, o por el subprograma, caso de no haberlas.

10.1. Pasos para el diseño de interfaz

Se ha de crear la lista de objetos que deberá intercambiar el módulo que llama y el subprograma llamado: necesidades, devoluciones y necesidades/devoluciones, marcando cada elemento como de entrada, salida o entrada/salida, respectivamente. A continuación, deberá estudiarse la conveniencia de diseñar el subprograma como función o como procedimiento. Por último, se determinará el paso de parámetros apropiado, señalando qué parámetros se van a pasar por valor y cuáles por referencia.

Habrá que establecer las precondiciones de uso del subprograma, que representan los requisitos que deben cumplirse antes de realizar la llamada, para que el subprograma funcione correctamente y, también, las postcondiciones, que especifican los asertos tras la ejecución del subprograma. Tenga en cuenta que las precondiciones van asociadas a las entradas del subprograma, mientras que las postcondiciones lo están a las salidas, por lo que se prestará especial atención en esta fase del diseño. Esto quiere decir que el programa que realiza la llamada tiene la responsabilidad de comprobar que los parámetros reales cumplen las especificaciones de la entrada y deberá saber que condiciones son ciertas cuando se termine la ejecución.

Observe que para el diseño de la interfaz no se necesita la definición real del subprograma aunque, como es lógico, finalmente deberá realizarse su definición para que sea posible su compilación y ejecución. Tenga en cuenta que el código generado deberá ser congruente con la interfaz establecida.

En ningún caso, deberá pasarse directamente a la escritura del código del subprograma sin haber estudiado con anterioridad la interfaz. Aunque esta es una costumbre entre ciertos programadores, es un foco de futuros errores y pérdida de tiempo y de calidad. Por otro lado, el programador no siempre decide dicha interfaz, es probable que le venga determinada por otros programadores o analistas en el transcurso del desarrollo de aplicaciones, por lo que deberá aprender a generar el código a partir de las interfaces suministradas.

También, deberá tener en cuenta que, a veces, se está interesado en la construcción de librerías de funciones y procedimientos de uso público, por lo que la interfaz supondrá la documentación externa de los subprogramas, cuyo objetivo es hacer posible el correcto uso por parte de otros programadores en otras aplicaciones.

10.2. ¿Cómo queda el diseño del algoritmo?

ANÁLISIS

A todo lo incluido hasta ahora, habrá que añadir los puntos característicos de la construcción de subprogramas.

- **Programa principal generalizado**. Dará una visión clara de la estructura general del programa, si es iterativo, si contiene distintas opciones, si estas son también iterativas, etc. El pseudocódigo generalizado incluye básicamente las llamadas a todos los módulos de primer nivel y algunas otras sentencias, generalmente de refuerzo de código.

- **Estudio y diseño de la interfaz de cada subprograma**. Se decidirá que módulos se convierten en subprogramas y se realizará su estudio de interfaz y la documentación. Cuando se implemente en C el subprograma, será conveniente escribir el diseño de interfaz junto a la definición de cada subprograma, como documentación añadida al diseño.

- **Diagrama jerárquico de estructura de módulos y/o de descomposición funcional**. Puede realizarse el DEM y señalar con * aquellos módulos que se convierten en funciones o procedimientos o, bien, realizar el DDF.

PSEUDOCÓDIGO

```
ENTORNO
   Constantes
   Variables
   Procedimientos
      nombreProc (listaTipos)  //Declaración de cabecera de procedimientos
   Funciones
      TipoA nombreFunción(listaTipos) //Declaración de cabecera de funciones

   Programa Principal
   <Inicio>
      <Declaraciones de variables y otros objetos locales del PP, entre ellos la
       siguiente: >
       TipoA var
      .....
      nombreProc (lista-Par_reales-proc)  //llamada a procedimiento
      .....
      var = nombreFunción (lista-Par_reales-func) //llamada a función
      .....

   <FinPP>
```

DEFINICIONES DE SUBPROGRAMAS

```
   nombreProc (lista-declaración par_formales) /*cabecera del subprograma
                                                 procedimiento*/
   <Inicio>
      <Declaración de objetos locales>
      .....
      .....
   <Fin-nombreProc>

   TipoA nombreFuncion (lista-declaración par_formales) /*cabecera del
                                                          subprograma
                                                          función*/
   <Inicio>
      <Declaración de objetos locales, entre ellos el siguiente: >
      TipoA nombreVariable
          .....
          .....
      Devolver (nombreVariable)
   <Fin-nombreFuncion>
```

Con la sentencia *Devolver* se asocia al nombre de la función el valor de *nombreVariable*, que será el valor devuelto al programa principal.

TRAZA Y REVISIÓN

Se han de realizar las siguientes actividades:

- La traza y revisión de cada módulo por separado.
- La traza y las oportunas revisiones del programa principal, una vez comprobado que los subprogramas funcionan. Si hay paso por referencia, es conveniente realizar la traza en ambos sentidos.
- La traza de integración, haciéndola paso a paso siguiendo el control de flujo del programa, tal como se haría en ejecución.

11. LLAMADAS A UN SUBPROGRAMA DESDE DISTINTOS MÓDULOS

Se ha tratado la reutilización de código como una de las razones esgrimidas para el uso de la programación modular, además de otras razones en cuanto a facilidad de diseño, legibilidad, etc. Si en un programa existe una tarea que se repite en diferentes puntos, conviene diseñarla como subprograma interno y llamarlo desde esos distintos lugares. Si por otro lado, alguna tarea en particular no sólo se repite varias veces en un programa, sino que además, es frecuente su aparición en otros programas, conviene convertirla en subprograma externo, para que pueda ser usado por cualquier aplicación, configurando para ello las librerías de usuario y contribuyendo directamente a la reutilización del código ya escrito.

ALGUNAS RECOMENDACIONES

En el diseño y uso de subprogramas es importante prestar especial atención a los siguientes puntos:

- Definir cuidadosamente la interfaz de comunicación.
- Declarar los procedimientos y funciones antes de que se utilicen en una llamada.
- Comprobar los tipos de datos de los parámetros formales y reales.
- Verificar el emparejamiento de parámetros formales y reales, tanto en número como en orden.
- Confirmar la llamada según sea función o procedimiento:
    - El nombre de una función deberá aparecer a la derecha de un símbolo de asignación. Bien asignando el nombre de la función a una variable

que sea del mismo tipo que el valor devuelto por la función o bien puede aparecer como parte de una expresión compleja.

- El nombre de un procedimiento no puede aparecer a la derecha de un símbolo de asignación.

CUANDO UTILIZAR FUNCIONES Y PROCEDIMIENTOS

No existen normas explícitas acerca de cuando deben diseñarse los subprogramas como funciones y cuando cómo procedimientos, pero se pueden seguir algunos criterios al respecto:

1. Si el subprograma no tiene salidas o tiene más de una, es aconsejable utilizar un procedimiento.

2. Si sólo tiene una salida, es recomendable usar una función.

3. Si tiene una sola salida, que además es entrada, es decir, es un parámetro de entrada/salida y, por tanto, debe pasarse por referencia, se usará un procedimiento.

4. Si para el diseño del subprograma ambos tipos son aceptables, se puede utilizar aquella que más guste al diseñador.

EJERCICIO PRÁCTICO DE APLICACIÓN

Se aplicarán los nuevos conocimientos al ejercicio desarrollado al comienzo de la unidad, completando los puntos referentes al diseño de subprogramas.

Análisis. Se incorpora al análisis realizado anteriormente lo siguiente:

Estudio y **DISEÑO DE INTERFAZ** de cada subprograma.

En primer lugar, se planteará cuáles de los módulos que aparecen en el algoritmo generalizado o en los siguientes niveles, es conveniente convertirlos en subprogramas. Se ha aconsejado transformar en funciones o procedimientos aquellas tareas que tengan consistencia de programa completo.

En concreto, al hacer el cálculo explícito del factorial y el múltiplo, se ve que tienen ambos categoría de funciones, puesto que realizan cálculos matemáticos y los resultados que producen están basados en esos cálculos, además, pueden devolverlos asociados a su nombre. Sin embargo, calcular la tabla de multiplicar, aunque realiza una operación matemática, debería devolver demasiados valores al programa principal y eso no es posible. Como aún no se conoce una estructura de datos que resuelva este problema, se diseñará como procedimiento.

Por otro lado, la presentación del menú de opciones parece un módulo idóneo para convertirlo en procedimiento, puesto que no realiza cálculos y asociado a su nombre es evidente que no debe devolver nada al programa principal.

Procedimientos

MENU

Proceso: Presenta en pantalla un menú de opciones

1. Listado de necesidades y devoluciones.

   - Necesidades: No tiene → Entradas: No tiene

   - Devoluciones: No tiene → Salidas: No tiene

     Sólo presenta mensajes en pantalla y los mensajes no son datos de salida.

   - Necesidades/devoluciones: No tiene → Entradas/salidas: No tiene

   - Precondiciones: No tiene

   - Postcondiciones: No tiene

2. Decidir si se diseña como función o procedimiento: Procedimiento.

3. ¿El paso de parámetros se hará por valor o referencia? No tiene parámetros.

NOTA: Esto es solamente la descripción del método de trabajo, por lo que cuando se termine el estudio se incluirá como documentación sólo entradas, precondiciones, salidas y postcondiciones.

Calcular_Mostrar_Tabla

Proceso: Calcula la tabla de multiplicar de un número y muestra en pantalla los resultados.

1. Listado de necesidades y devoluciones.

   - Necesidades: Un número → Entradas: Un número (como puede ser entero o real se decidirá más tarde).

   - Devoluciones: No tiene → Salidas: No tiene.

     Sólo muestra en pantalla los resultados.

   - Necesidades/devoluciones: No tiene → Entradas/salidas: No tiene.

   - Precondiciones: No tiene.

   - Postcondiciones: No tiene.

2. Decidir si diseñarlo como función o procedimiento: Procedimiento.

3. ¿El paso de parámetros se hará por valor o referencia? Como se trata de un parámetro de entrada se hará el paso por valor.

Funciones

FACTORIAL

Proceso: Calcula el factorial de un número.

1. listado de necesidades y devoluciones.

   - Necesidades: Un número → Entradas: Un entero (Numero) .
   - Devoluciones: Resultado de calcular el factorial → Salidas: Un entero.
   - Necesidades/devoluciones: No tiene → Entradas/salidas: No tiene.
   - Precondiciones: El número de la entrada debe ser entero positivo.
   - Postcondiciones: Asociado al nombre de la función se dispone del valor devuelto.

     Las postcondiciones aparecen totalmente definidas después de establecer claramente como se tratará la salida y esto no ocurrirá hasta decidir si se diseña el módulo como función o como procedimiento. Aunque en el texto está apareciendo ahora, en el estudio real se ha realizado antes el apartado 2 y el 3 que se detallan a continuación.

2. Se decide si diseñarlo como función o procedimiento: función.

3. ¿El paso de parámetros se hará por valor o referencia?: Como *Numero* es un parámetro de entrada se hará el paso por valor. Al haber elegido diseñar el módulo como función, que es lo adecuado, la salida se devolverá asociada al nombre, tal como se ha detallado en las postcondiciones.

   ¿Qué ocurriría si se hubiese decidido diseñar el módulo como procedimiento? En este caso, la salida tendría que devolverse por referencia y no aparecería su valor asociado al nombre del subprograma, aunque como postcondición se dispondría de su valor a través de la variable referenciada.

MULTIPLO3

Proceso: Calcula si un número es o no múltiplo de 3 devolviendo verdadero o falso.

1. listado de necesidades y devoluciones.

   - Necesidades: Un número → Entradas: Un entero (Numero).
   - Devoluciones: Un valor verdadero o falso → Salidas: Un carácter.

     El resultado verdadero o falso se indicará con un carácter, de manera que verdadero será 'S' y falso 'N'.

   - Necesidades/devoluciones: no tiene → Entradas/salidas: no tiene.
   - Precondiciones: No tiene.
   - Postcondiciones: Asociado al nombre de la función se obtiene el valor devuelto.

2. Se decide si diseñarlo como función o procedimiento: función.
3. ¿El paso de parámetros se hará por valor o referencia?: *Numero* es un parámetro de entrada, por tanto, su paso se hará por valor. La salida, como se ha elegido diseñar el módulo como función, será devuelta asociada al nombre, tal como se detalla en las postcondiciones.

Pseudocódigo

```
ENTORNO:
    /*Nombre del programa: MenuFMM3*/

    VARIABLES:
     Carácter: opcion, seguir
     Entero numero, factor, multiplo, resultado

    PROCEDIMIENTOS
        //Declaraciones de cabeceras de procedimientos
     MENU()
     Calcular_Mostrar_Tabla (entero)

    FUNCIONES
        //Declaraciones de cabeceras de funciones
     entero FACTORIAL (entero)
     carácter MULTIPLO3 (entero)
```

Programa principal detallado

```
 Inicio
  Repetir
    /*Mostrar MENU Elegir y Validar OPCION*/
         Repetir
            MENU() // Llamada a procedimiento sin parámetros
            Leer (opcion)
```

```
      Mientras (opcion < 1 o opcion > 4)
Según (opcion)
  Para opcion == 1
        /*Calcular Factorial*/
    Repetir
      /*Obtener y Validar NUMERO*/
      Repetir
        Escribir ("Introduzca un número positivo")
        Leer (numero)
      Mientras (numero < 0  O  numero > 100)
      factor = FACTORIAL (numero) //Llamada a función
      /*numero es el parámetro real
       factor es la variable donde se guarda el valor
       devuelto por la función.*/
      /*Mostrar Resultado Factorial*/
      Escribir("El factorial de --- es --- ", numero,
      factor)
      /*Leer y Validar SEGUIR*/
      Repetir
        Escribir ("Desea seguir (S/N)")
        Leer (seguir)
        Pasar A Mayúsculas SEGUIR
      Mientras (seguir != 'S'  Y seguir != 'N')
    Mientras (seguir == 'S')
  //fin opción 1
  Para opcion == 2
        /*Visualizar Tabla Multiplicar*/
    Repetir
      /*Obtener y Validar numero*/
      Repetir
        Escribir ("Introduzca un número positivo")
        Leer (numero)
      Mientras (numero < 0  o  numero > 100)
      Calcular_Mostrar_Tabla (NUMERO)/*Llamada a
                        procedimiento con parámetro*/
      /*Leer y Validar SEGUIR*/
      Repetir
        Escribir ("Desea seguir (S/N)")
        Leer (seguir)
        Pasar A Mayúsculas seguir
      Mientras (seguir != 'S'  Y seguir != 'N')
    Mientras (SEGUIR == 'S')
    //fin opción 2
  Para opcion == 3
        /*Comprobar  si es Múltiplo de 3*/
    Repetir
      /*Obtener y Validar NUMERO*/
      Repetir
```

```
                Escribir ("Introduzca un número positivo")
                Leer (numero)
              Mientras (numero < 0  O  numero > 100)
              MULTIPLO  = MULTIPLO3 (numero) /*Llamada a
                                        función con parámetro*/
              // numero es el parámetro real
              //multiplo guarda el valor devuelto
              /*Mostrar Resultado Múltiplo*/
              Si  (multiplo = = 'S')
                Escribir ("El número --- es múltiplo de 3",
                          numero)
              En otro caso
                Escribir ("El número --- no es múltiplo de 3",
                          numero)
              Finsi
              /*Leer y Validar SEGUIR*/
              Repetir
                Escribir ("Desea seguir (S/N)")
                Leer (seguir)
                Pasar A Mayúsculas SEGUIR
              Mientras (seguir != 'S'  Y seguir != 'N')
            Mientras (seguir == 'S')
            //fin opciòn 3
        Fin_según
     Mientras (opcion != 4)
    Fin PP

//DEFINICIÓN DE PROCEDIMIENTO
     Presentar MENU ( ) //No devuelve nada y no tiene parámetros
          Inicio
            Escribir ("  1- Calcular Factorial
                         2 –Visualizar tabla de multiplicar
                         3 – Comprobar si es múltiplo de 3
                         4 – Salir del programa ")
          Fin PM
```

Es importante considerar para la calidad del diseño que **los subprogramas no deben leer ni escribir**, salvo que sea ese su propósito. En el procedimiento que se diseña a continuación, nos hemos permitido la licencia de *escribir*, por no disponer, con los conocimientos actuales, de una estructura de datos adecuada para mandar al programa principal tantos valores como genera la tabla. Lo más acertado sería diseñarlo como función, pero para ello se necesitará saber utilizar las estructuras de datos apropiadas.

```
     Calcular y Mostrar Tabla (entero num) /*No devuelve nada,
                                      pero sí tiene parámetro */
          Inicio
            Entero contador, resultado
```

```
            Para (contador = 1; mientras contador <= 10;
                  contador = contador + 1)
              resultado = num * contador
              Escribir ("--- * --- = ---", num, contador,
              resultado)
            FinPara
        Fin CMT

//DEFINICIÓN DE FUNCIONES
    entero FACTORIAL(entero num) /*Devuelve un entero y tiene
                                   un parámetro*/
        Inicio
          Entero FACT, Contador
            Para (contador = num; mientras contador > 1;
                   contador = contador - 1)
              fact = fact * contador
            FinPara
          Devolver fact /*devuelve el valor que se asocia al
                         nombre*/
    Fin F
    carácter MULTIPLO3 (entero num) /*Devuelve un carácter y
                                      tiene un parámetro*/
        Inicio
          Carácter multiplo
           /*Esta variable es local de MULTIPLO3, no tiene
             nada que ver con la declarada en el programa
             principal como local de main*/
          Entero resto
          resto = num % 3
          Si (resto == 0)
             multiplo = 'S'
          En otro caso
             multiplo = 'N'
          FinSi
          Devolver multiplo /*devuelve el valor que se asocia
                             al nombre*/
    Fin M3
```