Sistemas y Aplicaciones Informáticas

Tema 27. Programación Estructurada. Estructuras Básicas. Funciones y Procedimientos.

1. Ámbito de docencia.

- Sistemas informáticos monousuario y multiusuario (ASI 1).
- Sistemas informáticos multiusuario y en red (DAI 1).
- Sistemas operativos en entornos monousuario y multiusuario (ESI 1).

2. Programación estructurada.

2.1. Concepto. Segmentación y estructura arborescente.

- La programación estructurada es un conjunto de técnicas para desarrollar algoritmos fáciles de escribir, verificar, leer y modificar, utilizando un diseño descendente del programa, unas estructuras de control limitadas y un ámbito limitado de las estructuras de datos del programa.
- Un programa estructurado esta compuesto de segmentos, que reflejan la división del programa en partes relacionadas entre sí en forma jerárquica, formando una estructura de árbol.
- El segmento superior de la estructura jerárquica, o programa principal, contiene las funciones de control de más alto nivel, mientras que los segmentos inferiores contienen funciones detalladas. Cualquier segmento puede ser sustituido por su descomposición arborescente y viceversa.
- Una segmentación bien diseñada deberá mostrar las relaciones existentes entre las distintas funciones de manera que sea fácil comprender lo que debe hacer el programa y asegurar que efectivamente lo realice. Este hecho garantizará que los cambios que se efectúen a una parte del programa no afecten al resto del programa que no ha sufrido cambios.

2.2. Técnicas utilizadas.

2.2.1. Diseño descendente. Abstracción funcional. Refinamiento.

- El diseño descendente consiste en dividir el problema en varios subproblemas que se pueden resolver por separado, para después recomponer los resultados y obtener la solución al problema.
- Los subproblemas se dividen a su vez en subproblemas más pequeños hasta que su solución pueda implementarse fácilmente, dando lugar a sucesivos segmentos. Se aplica la técnica de abstracción funcional, mediante la cual en cada descomposición se supone que los subproblemas más pequeños están resueltos, dejando su realización al siguiente nivel de descomposición.
- Posteriormente se realiza el refinamiento de los segmentos finales mediante la descripción de los pasos a llevar a cabo. Para que el refinamiento sea correcto hay que definir con precisión el cometido de cada segmento, garantizar su corrección, y ensamblar y organizar bien entre sí los subalgoritmos resultantes. El tamaño de un segmento depende de su función y de su aplicación.
- La utilización de esta técnica de diseño tiene los siguientes objetivos básicos:
  - *Simplificación*. Las diferentes partes del problema pueden ser diseñadas y desarrolladas de modo independiente y por diferentes personas.
  - *Segmentación*. El diseño final queda estructurado en forma de segmentos, lo que hace más sencilla su implementación y su posterior mantenimiento.

2.2.2. Programas propios. Teorema de la estructura.

- Cada uno de los segmentos de un programa es un programa propio si se cumple lo siguiente:
  - Tiene un único punto de entrada y un único punto de salida.
  - Existen caminos desde la entrada hasta la salida que pasan por todas las partes del segmento.

    - Todas las instrucciones son ejecutables y no existen bucles sin fin.
- El teorema de la estructura de Böhm y Jacopini establece que un programa propio puede ser escrito utilizando solamente las siguientes estructuras lógicas de control:
    - *Secuenciales*. Sucesión simple de dos o más operaciones.
    - *Alternativas*. Ejecución condicional de una o más operaciones.
    - *Iterativas*. Repetición de una operación mientras se cumple una condición.
- Cuando varios programas propios se combinan utilizando las tres estructuras básicas de control mencionadas anteriormente, el resultado es también un programa propio.

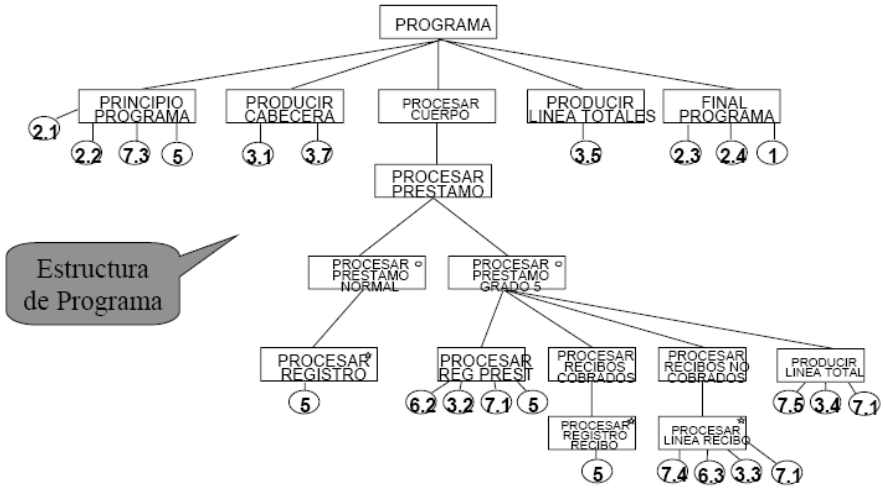
2.3. Características de calidad en los programas.

- **Corrección**. Deben resolver los problemas para los que están diseñados.
- **Comprensibilidad**. Incluye la legibilidad y la buena documentación, características que permiten una mayor facilidad y comodidad en el mantenimiento de los programas.
- **Eficiencia**. Expresa los requerimientos de memoria y el tiempo de ejecución del programa.
- **Flexibilidad**. Capacidad de adaptación del programa a variaciones del problema inicial, lo cual permite la utilización del programa durante mayor tiempo.
- **Portabilidad**. Posibilidad de usar el mismo programa sobre distintos sistemas sin realizar cambios notables en su estructura.

2.4. Métodos de representación.

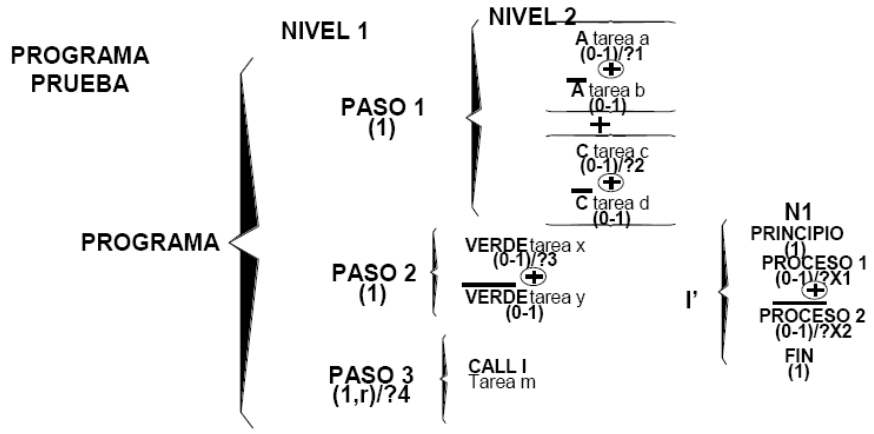
2.4.1. Jackson. Representación.

- Se basa en el principio de que la base inicial del diseño del programa son los datos del problema y no los requisitos funcionales exigidos. Tiene las siguientes características:
    - Permite una mayor objetividad.
    - Es necesario partir de una buena especificación del problema a resolver, conociendo los datos de entrada, los datos de salida y los algoritmos aplicables.
    - Una vez obtenida una estructura objetiva del problema, que constituye un reflejo del mundo real con el que trata el programa, resulta más fácil asignar las distintas funciones a realizar.
- Las fases del modelo de Jackson son las siguientes:
    - Formar las estructuras de datos de entrada y de salida a partir de los datos del problema.
    - Determinar las correspondencias o los elementos comunes entre ambas estructuras de datos.
    - En función de las correspondencias obtener una estructura única para el programa, que puede traducirse fácilmente a un diagrama de flujo de control.
    - Asignar a la estructura del programa las operaciones ejecutables de programa derivadas de las especificaciones funcionales.
    - Traducir el conjunto estructura-operaciones a un formato de pseudocódigo cuya codificación resulte sencilla.
    - Las estructuras de datos de entrada y salida y la estructura del programa se documentan mediante diagramas de estructura de Jackson, los cuales contienen las estructuras secuenciales, alternativas e iterativas.
- Un diagrama de estructura de Jackson tiene el siguiente aspecto:



2.4.2. Warnier. Representación.

- Se basa en la aplicación de los siguientes principios:
  - La ordenación jerárquica de los conjuntos de información (salida, entrada y programa).
  - La organización jerárquica de los datos de entrada determinada por los datos de salida.
  - La organización del programa viene determinada por los datos de entrada.
  - El control del programa se realiza a partir de los datos de salida.
- Las fases del modelo de Warnier son las siguientes:
  - Determinación de la estructura de los datos de salida.
  - Determinación de la estructura de los datos de entrada en función de la salida deseada.
  - Determinación de la estructura óptima del programa basada en la estructura de entrada.
  - Creación de una lista de pseudoinstrucciones.
  - Asignación de las mismas a cada elemento de la estructura del programa.
- El modelo de Warnier se representa mediante:
  - Secuencia de diversos elementos que se suceden de arriba a abajo en un mismo nivel.
  - Repetición de ocurrencias dentro de un mismo conjunto, que se representan en los diagramas indicando el número mínimo y máximo de las mismas. Por ejemplo (0, n).
  - Selección entre ocurrencias de un conjunto, que efectúa la subdivisión en subconjuntos excluyentes entre sí cuya presencia es aleatoria, y se representa por medio del símbolo +.



3. Estructuras básicas.

3.1.1. Secuencial.

- Consiste en la ejecución ordenada de una serie de primitivas, empezando por la primera que se encuentre en el código y acabando por la última.
- Una primitiva es una acción sobre los datos que se lleva a cabo inmediatamente, y puede ser una:
  - *Asignación*. Consiste en pasar a una variable un determinado valor, que puede ser una constante o el resultado de una expresión, compatible con el tipo de dato de la variable.
  - *Entrada de datos*. Consiste en recibir desde un dispositivo de entrada (por ejemplo el teclado) un determinado valor, y almacenarlo en una variable.
  - *Salida de datos*. Consiste en enviar a un dispositivo de salida (un monitor o una impresora) el contenido de una variable, un resultado o un mensaje.

3.1.2. Alternativa.

- Se basa en la evaluación de una expresión lógica o relacional para controlar la ejecución de ciertas instrucciones y alterar de este modo el orden secuencial del programa.
- Existen tres tipos básicos de estructuras de control alternativas:
  - *Simples*. Si la expresión devuelve un valor lógico verdadero, se lleva a cabo un conjunto de instrucciones. En caso contrario no se ejecuta y se continúa con la siguiente instrucción.
  - *Dobles*. Si la expresión devuelve un valor lógico verdadero, se lleva a cabo un conjunto de instrucciones. En caso contrario se ejecuta otro conjunto diferente de instrucciones.
  - *Múltiples*. Controla la ejecución de varios conjuntos de acciones mediante el valor final de una expresión, de tal forma que cada conjunto de acciones está ligado a un posible valor de la expresión. Normalmente existe un bloque que especifica las instrucciones a llevar a cabo en caso de que la expresión tenga algún valor no definido.

3.1.3. Iterativa.

- Se basa en la evaluación de una expresión lógica o relacional para repetir la ejecución de ciertas instrucciones y alterar de este modo el orden secuencial del programa.
- Existen tres tipos básicos de estructuras de control iterativas:
  - *Mientras*. Primero se evalúa la expresión. Si devuelve un valor lógico verdadero, se ejecuta un conjunto de instrucciones y se vuelve a evaluar la expresión, repitiéndose la ejecución de las instrucciones hasta que la expresión tenga un valor lógico falso. Si en la primera evaluación el valor de la expresión es falso, el conjunto de instrucciones no se ejecuta.
  - *Repetir hasta*. Primero se ejecuta un conjunto de instrucciones y después se evalúa la expresión. Si devuelve un valor lógico falso, se repite la ejecución de las instrucciones y la evaluación de la expresión hasta que ésta tenga un valor lógico verdadero. En cualquier caso el conjunto de instrucciones se ejecuta al menos una vez.
  - *Para-Hasta*. Repite un conjunto de instrucciones un cierto número de veces en función del contenido de una variable de control. En primer lugar se le asigna un determinado valor y después se evalúa una expresión en la que participe dicha variable. Si devuelve un valor lógico verdadero, se ejecuta un conjunto de instrucciones, se actualiza el valor de la variable y se vuelve a evaluar la expresión, repitiéndose la ejecución de las instrucciones y la

actualización de la variable hasta que la expresión tenga un valor falso. Si en la primera evaluación el valor de la expresión es falso, el conjunto de instrucciones no se ejecuta.

4. Funciones y procedimientos.

4.1. Concepto de procedimiento y función.

- Los procedimientos y las funciones son segmentos de programa que forman parte de la resolución independiente de los subproblemas resultantes de la descomposición de un problema. Completan y amplían el diseño descendente como método de resolución de problemas.
- Un procedimiento es un conjunto de instrucciones identificado por un nombre que realiza una determinada tarea dentro de un programa. Este conjunto de instrucciones puede ser activado mediante una llamada desde otra parte del programa o desde otro procedimiento.
- Una función es un procedimiento que siempre devuelve un valor de un determinado tipo, sea cual sea el camino de ejecución seguido. Por este motivo una llamada a una función es una expresión válida, y puede formar parte de expresiones más complejas.
- La definición de ambos aparece en la parte de declaraciones de un programa, y consiste en:
  - *Una cabecera*, que contiene el nombre del procedimiento o la función, seguido del nombre y el tipo de dato de cada uno de los parámetros. En el caso de una función, debe incluir además el tipo de dato del valor devuelto.
  - *Un cuerpo*, formado por una parte de declaraciones de variables locales y una parte de instrucciones que indican las acciones a realizar. En el caso de una función, debe incluir una instrucción que le asigne algún valor de retorno.

4.2. Parámetros.

4.2.1. Parámetros actuales y formales. Correspondencias.

- Los datos externos que necesita un procedimiento o una función para realizar sus tareas se pasan a través de un número limitado de parámetros en el momento de la llamada. Cada parámetro está definido por un nombre y un tipo de dato asociado.
- En la definición de un procedimiento o una función, los datos genéricos sobre los que deben actuar se expresarán mediante una lista de parámetros formales. En la llamada, los datos concretos sobre los que actúan se expresarán mediante una lista de parámetros actuales.
- Al llamar un procedimiento o una función, se produce una correspondencia entre los parámetros formales y los parámetros actuales:
  - *Correspondencia posicional*. Se emparejan según la posición que tengan en la definición del procedimiento o de la función, y en la llamada realizada respectivamente. Esto obliga a que coincidan el número de parámetros y el tipo de dato de cada asociación actual-formal.
  - *Correspondencia por nombre*. La correspondencia entre los parámetros formales y los actuales se indica explícitamente en la llamada al procedimiento o a la función.

4.2.2. Tipos de parámetros.

- Cuando termina la ejecución de un procedimiento o una función, se puede conseguir que las modificaciones realizadas sobre los parámetros formales se reflejen sobre los correspondientes parámetros actuales de la llamada.

- Bajo esta consideración, se puede hablar de tres tipos de parámetros:
  - *Parámetros de entrada*. El parámetro actual de la llamada tiene un valor definido, que además es el valor que tomará inicialmente el correspondiente parámetro formal. Si al finalizar la acción no interesa que las posibles modificaciones del parámetro formal se reflejen sobre el parámetro actual asociado, deberá definirse como de entrada.
  - *Parámetros de salida*. El parámetro actual de la llamada no tiene un valor definido. Si al finalizar la acción interesa que las posibles modificaciones del parámetro formal se reflejen sobre el parámetro actual asociado, deberá definirse como de salida.
  - *Parámetros de entrada/salida*. El parámetro actual de la llamada tiene un valor definido, que además es el valor que tomará inicialmente el correspondiente parámetro formal. Si al finalizar la acción interesa que las posibles modificaciones del parámetro formal se reflejen sobre el parámetro actual asociado, deberá definirse como de entrada/salida.

4.2.3. Paso de parámetros.

- **Paso por valor**. Los parámetros formales declarados se inicializan con los valores de los parámetros actuales. De esta forma, se trabaja con una copia de los parámetros actuales que desaparece al finalizar el procedimiento o la función. Cualquier modificación de los parámetros formales no se refleja en los correspondientes parámetros actuales.
- **Paso por valor resultado**. Los parámetros formales reciben una copia de los valores de los parámetros actuales y al finalizar la ejecución del subprograma se realiza el proceso inverso.
- **Paso por referencia**. Los parámetros formales declarados se inicializan con la dirección de los parámetros actuales. De esta forma, la dirección en memoria del parámetro formal coincide con la del parámetro real y cualquier referencia a dicho parámetro formal en el interior del procedimiento o de la función realmente se refiere al contenido de esa dirección. Indirectamente se puede llegar a modificar el valor original del parámetro actual.

4.3. Ámbitos.

4.3.1. Estructura de bloques. Ámbito de procedimientos y funciones.

- El programa principal, los procedimientos y las funciones en él declarados, y los que a su vez pudieran declararse dentro de ellos, constituyen un conjunto de bloques anidados que determinan el ámbito de validez de las variables y de los procedimientos y funciones.
- El ámbito de un procedimiento o una función es el conjunto de todos aquellos procedimientos o funciones que pueden llamarle. Si un procedimiento o una función está declarado en otro procedimiento u otra función, se dice que el primero es descendiente del segundo, y que el segundo es ascendiente del primero.
- En general, el ámbito de un procedimiento o una función comprende lo siguiente:
  - Los ascendientes pueden llamar a sus descendientes, pero los ascendientes indirectos no pueden llamar a descendientes indirectos.
  - Cualquier procedimiento o cualquier función puede llamarse a sí mismo.
  - Un procedimiento o una función puede llamar a todos los descendientes directos del principal o del ascendiente directo que hayan sido declarados con anterioridad.

4.3.2. Variables locales y globales. Efectos laterales y precedencia de nombre.

- Las variables locales son aquellas que sólo tienen significado dentro del procedimiento o de la función en que están definidas y no son visibles fuera de él.
- Las variables globales son aquellas que están declaradas fuera de cualquier procedimiento o función y que son visibles en todo el programa.
- Debe tenerse en cuenta lo siguiente:
  - Todas las variables usadas por un procedimiento o una función que no sean variables globales deben declararse como variables locales de ese procedimiento o esa función.
  - Los parámetros de un procedimiento o una función actúan como variables locales.
  - La información que se pasa entre procedimientos y funciones debe hacerse a través de una lista de parámetros y no a través de variables globales. Esto hace que sean más independientes y facilita tanto el diseño como la prueba de cada uno de ellos.
- Cuando se modifica una variable global en un procedimiento o una función, se dice que se produce un efecto lateral. Los efectos laterales deberán evitarse pues introducen dependencias indeseables entre procedimientos y funciones.
- En caso de que una variable local y una variable global tengan el mismo identificador, cuando se haga una referencia a éste siempre prevalecerán los datos que presenten un ámbito más reducido, es decir, las variables locales frente a las variables globales (precedencia de nombre).

4.4. Llamadas a procedimientos y funciones.

4.4.1. Proceso de llamada. Registros de activación.

- Para ejecutar un programa se reserva una zona de memoria principal donde se almacenan las variables y constantes globales. Además cada programa dispone de una zona de memoria diferenciada para trabajar con procedimientos y funciones, denominada pila de ejecución.
- Para gestionar el proceso de llamada y retorno se utilizan los registros de activación:
  - Un registro de activación es un espacio de memoria situado en la pila que se crea en tiempo de ejecución cuando se llama a un procedimiento o una función. Contiene los parámetros y las variables locales del procedimiento o la función, e información de control para volver a procedimiento llamador después de la llamada.
  - Cada registro de activación se referencia por su dirección de inicio. En cada momento sólo un registro de activación está activo, el correspondiente al procedimiento o a la función actual, que está situado en la cima de la pila y es apuntado por un registro de la CPU.
- Las llamadas a sucesivos procedimientos y funciones hace que se vayan apilando los registros de activación de cada llamada. Al finalizar la ejecución de un procedimiento o una función, se recupera la información de su registro de activación y libera su espacio, apuntando al siguiente elemento de la pila, que corresponde al registro de activación del procedimiento llamador.
- Por tanto, no se continúa con la ejecución de un procedimiento o una función hasta que no finalice el procedimiento llamado. Esta continuación es posible ya que en el registro de activación del procedimiento llamado se guarda la dirección de la siguiente instrucción a ejecutar después de la llamada.

4.4.2. Secuencias de llamada y retorno.

- Las llamadas a procedimientos se implementan mediante la generación de secuencias de llamada y secuencias de retorno. Incluyen el código adicional encargado de la gestión de los registros de activación. Las tareas se reparten entre el procedimiento llamador y el procedimiento llamado, y son muy dependientes de la arquitectura.
- **Secuencia de llamada**. Crea el registro de activación del nuevo procedimiento y asigna sus valores correspondientes (parámetros y estado).
  - *Procedimiento llamador*:
    - Crea el nuevo registro de activación, guarda el valor del puntero al registro de activación actual en él, y establece dicho puntero hacia el nuevo registro de activación.
    - Evalúa los parámetros actuales y los guarda en el nuevo registro de activación.
    - Calcula dirección de la siguiente instrucción a ejecutar después de la llamada y la guarda en el nuevo registro de activación.
    - Salta a la primera instrucción del procedimiento llamado.
  - *Procedimiento llamado*:
    - Guarda valores registros de la CPU y la información de estado.
    - Inicializa las variables locales del procedimiento llamado e inicia su ejecución.
- **Secuencia de retorno**. Restablece el estado de la máquina para que el procedimiento llamador continúe su ejecución.
  - *Procedimiento llamado*:
    - Coloca el valor devuelto al inicio de su registro de activación si se trata de una función.
    - Restablece el estado de la CPU. El puntero a la pila de registros de activación vuelve a apuntar al registro de activación llamador.
  - *Procedimiento llamador*:
    - Se recupera la ejecución del procedimiento llamador directamente, al restaurar el contador de programa, o cargando explícitamente la dirección de retorno guardada en el registro de activación.
    - Continúa su ejecución, ya que el valor devuelto aún está en la dirección a continuación de su registro de activación.