Modularización

Uso de subprogramas

Razones válidas para crear un subprograma
Cohesión y acoplamiento
Pasos para escribir un subprograma
El nombre y los parámetros de un subprograma

Tipos de datos abstractos (TDAs)

Bibliografía

- ✓ Steve McConnell: “Code Complete”.
  Estados Unidos: Microsoft Press, 1994. ISBN 1-55615-484-4.

Uso de subprogramas

Razones válidas para crear un subprograma

- Reducir la complejidad del programa (“divide y vencerás”).
- Eliminar código duplicado.
- Limitar los efectos de los cambios (aislar aspectos concretos).
- Ocultar detalles de implementación (p.ej. algoritmos complejos).
- Promover la reutilización de código (p.ej. familias de productos).
- Mejorar la legibilidad del código.
- Facilitar la portabilidad del código.

Cohesión

Medida del grado de identificación de un módulo con una función concreta.

Cohesión aceptable (fuerte)

- Cohesión funcional (un módulo realiza una única acción).
- Cohesión secuencial (un módulo contiene acciones que han de realizarse en un orden particular sobre unos datos concretos).
- Cohesión de comunicación (un módulo contiene un conjunto de operaciones que se realizan sobre los mismos datos).
- Cohesión temporal (las operaciones se incluyen en un módulo porque han de realizarse al mismo tiempo; p.ej. inicialización).

Cohesión inaceptable (débil)

- Cohesión procedural (un módulo contiene operaciones que se realizan en un orden concreto aunque no tengan nada que ver entre sí).
- Cohesión lógica (cuando un módulo contiene operaciones cuya ejecución depende de un parámetro: el flujo de control o “lógica” de la rutina es lo único que une a las operaciones que la forman).
- Cohesión coincidental (cuando las operaciones de una rutina no guardan ninguna relación observable entre ellas).

Acoplamiento

Medida de la interacción de los módulos que constituyen un programa.

Niveles de acoplamiento (de mejor a peor):

- Acoplamiento de datos (acoplamiento normal): Todo lo que comparten dos rutinas se especifica en la lista de parámetros de la rutina llamada.

- Acoplamiento de control: Una rutina pasa datos que le indican a la otra rutina qué hacer (la primera rutina tiene que conocer detalles internos de la segunda).

- Acoplamiento externo: Cuando dos rutinas utilizan los mismos datos globales o dispositivos de E/S.

  Si los datos son de sólo lectura, el acoplamiento se puede considerar aceptable. En general, este tipo de acoplamiento no es deseable porque la conexión existente entre los módulos no es visible.

- Acoplamiento patológico: Cuando una rutina utiliza el código de otra o altera sus datos locales ("acoplamiento de contenido").

  La mayor parte de los lenguajes estructurados incluyen reglas para el ámbito de las variables que impiden este tipo de acoplamiento.

Objetivo
Reducir al máximo el acoplamiento y aumentar la cohesión de los módulos.

Pasos para escribir un subprograma

1. Definir el problema que el subprograma ha de resolver.
2. Darle un nombre no ambiguo al subprograma.
3. Decidir cómo se puede probar el funcionamiento del subprograma.
4. Escribir la declaración del subprograma (cabecera de la función).
5. Buscar el algoritmo más adecuado para resolver el problema.
6. Escribir los pasos principales del algoritmo como comentarios.
7. Rellenar el código correspondiente a cada comentario.
8. Revisar mentalmente cada fragmento de código.
9. Repetir los pasos anteriores hasta quedar completamente satisfecho.

El nombre de un subprograma

- Procedimiento: Verbo seguido de un objeto.
- Función: Descripción del valor devuelto por la función.

✓ El nombre debe describir todo lo que hace el subprograma.

✓ Se deben evitar nombres genéricos que no dicen nada (p.ej. `calcular`)

✓ Se debe ser consistente en el uso de convenciones.

Los parámetros de un subprograma

✓ Orden: (por valor, por referencia) == (entrada, entrada/salida, salida)

✓ Si varias rutinas utilizan los mismos parámetros, éstos han de ponerse en el mismo orden (algo que la biblioteca estándar de C no hace).

✓ De acuerdo con la primera norma, las variables de estado o error se ponen al final.

✓ No es aconsejable utilizar los parámetros de una rutina como si fuesen variables locales de la rutina.

✓ Se han de documentar las suposiciones que se hagan acerca de los posibles valores de los parámetros.

✓ Sólo se deben incluir los parámetros que realmente necesite la rutina para efectuar su labor.

✓ Las dependencias existentes entre distintos módulos han de hacerse explícitas mediante el uso de parámetros.

Tipos de datos abstractos
TDAs

Los tipos definidos por el usuario son una de las capacidades más importantes que ofrecen los lenguajes de programación para clarificar el significado de un programa.

- Se simplifican las modificaciones que tengamos que realizar.

- Se encapsulan detalles de implementación (en vez de aparecer por todas partes, estos detalles sólo aparecen en la declaración del tipo).

- Se amplía el “vocabulario” que podemos utilizar en nuestro programas.

TDA

Colección de datos y todas las operaciones que se efectúan sobre esos datos

Ejemplos: Una ventana, un menú, un fichero, una lista, una tabla...

PUNTO CLAVE: El acceso a los datos se realiza siempre a través de las operaciones definidas en el propio TDA.

Beneficios de la utilización de TDAs

- Se ocultan detalles de implementación.
- Se facilitan las modificaciones que puedan afectar al programa.
- Es más fácil mejorar la eficiencia del programa.
- Es más fácil verificar la corrección del programa.
- La legibilidad de los programas mejora.
- Se simplifican las interfaces de los módulos del programa.
- Se definen operaciones en función de los objetos sobre los que se efectúan.