ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Universidade Vigo

Programación I
Tema 3

Grado en Ingeniería Informática

María José Lado Touriño - Xosé Antón Vila Sobrino

Tema 3. Técnicas de Diseño de Algoritmos

1. Estructura General de un Algoritmo
2. Diseño Descendente: Programación Estructurada
3. Diseño Modular

1. Estructura General de un Algoritmo

- Necesario considerar:
  - **Cabecera**:
    - Nombre del algoritmo precedido de la palabra ALGORITMO.
  - **Cuerpo:** dos bloques:
    - **Declaraciones:** se definen o declaran constantes, variables, tipos de datos definidos por el usuario, variables.
    - **Instrucciones:**
      - Inicio/fin: primera (INICIO) y última (FIN) instrucciones.
      - Entrada de datos desde el dispositivo estándar: LEER(<lista_de_variables>)
      - Salida de datos en el dispositivo estándar: ESCRIBIR(<lista_de_expresiones>).
      - Asignación: <nombre_de_variable> ← <valor>.
      - Bifurcación: se controla que se ejecuten o no las instrucciones, y se altera el orden de ejecución de las mismas.
      - Comentarios: de una sola línea *//* o de varias líneas **{}**.

Contadores

- Su valor se incrementa o decrementa cada vez que se produce una determinada acción.
- Se suelen utilizar en estructuras repetitivas para contar acciones internas del bucle.
- Inicialización (asignación de valor inicial).
- Incremento o decremento (se colocan en el interior del bucle).
- Ejemplo:

```
VARIABLES
    contador, x : Entero
ALGORITMO CONTAR_PARES
    contador <- 0
    DESDE x <- 1 HASTA 10
        SI (x MOD 2 = 0)
            contador <- contador + 1
        FIN_SI
        FIN_DESDE
    ESCRIBIR("El número de valores pares es ", contador)
FIN
```

Acumuladores

- Su valor se incrementa o decrementa en una cantidad variable.
- Inicialización.
- Incremento o decremento.
- Ejemplo:

```
VARIABLES
    acumulador, x : Entero
ALGORITMO SUMAR_PARES
    acumulador <- 0
    DESDE x <- 1 HASTA 10
        SI (x MOD 2 = 0)
            acumulador <- acumulador + x
        FIN_SI
    FIN_DESDE
    ESCRIBIR("La suma de valores pares es ", acumulador)
FIN
```

2. Diseño Descendente: Programación Estructurada

- Paradigma de programación para desarrollar algoritmos con claridad, fáciles de escribir, verificar, leer y modificar.
- Emplea:
  - Diseño descendente (top-down): diseño de algoritmos en etapas, de los conceptos generales a los detalles.
  - Recursos abstractos: descomposición de tareas complejas suponiendo que cada una de sus partes ya está resuelta.
  - Estructuras básicas: se escriben los algoritmos con las estructuras:
    - Secuenciales.
    - Condicionales.
    - Repetitivas.

Teorema de Böhm y Jacopini

- Programa propio: debe cumplir:
  - Tiene un único punto de entrada y otro de salida.
  - Existe siempre, al menos un camino que va desde el inicio hasta el fin del algoritmo, y se puede seguir.
  - No posee bucles infinitos.

Para que la programación sea estructurada todos los programas tienen que ser propios.

Un programa propio se escribe empleando únicamente estructuras secuenciales, selectivas y repetitivas.

Ventajas

- Control de ejecución: ejecución secuencial pero puede haber partes que se ejecuten o no en función de determinadas condiciones.
- Programas más fáciles de entender.
- Pueden ser leídos de forma secuencial.
- Estructura de los programas es clara.
- Reducción del esfuerzo en las pruebas y depuración.
- Los errores se pueden detectar y corregir más fácilmente.
- Reducción de los costes de mantenimiento.
- Programas más sencillos y más rápidos de confeccionar.
- Se incrementa el rendimiento de los programadores.

3. Diseño Modular

- Amplía el diseño descendente como método de resolución de problemas.
- División de un programa en partes más manejables (usualmente denominadas segmentos o módulos).
- Cada segmento tiene solamente una entrada y una salida, no posee bucles infinitos y no tiene instrucciones que nunca se ejecuten.
- La comunicación entre segmentos se lleva a cabo de manera controlada.
- Existe dependencia nula (o casi) entre los módulos: se puede trabajar con cada módulo de forma independiente.
- Los cambios que se efectúen en una parte del programa, no afectan al resto del programa que no ha sufrido cambios.

Módulo principal

- Transfiere el control a los distintos módulos o subalgoritmos.

Subalgoritmos o módulos

- Pequeños.
- Siguen las reglas de la programación estructurada.
- Se pueden representar con pseudocódigo.

Encapsulación

- Ocultación de detalles de las funciones contenidas en un módulo.

Ventajas

- Menor coste al resolver varios subproblemas de forma aislada, con respecto a la resolución del problema global.
- Como los módulos son independientes se puede trabajar en paralelo.
- Se puede modificar un subalgoritmo sin que esto afecte a los demás.
- Los subalgoritmos solo se escriben una vez, aunque se empleen varias veces a lo largo de todo el algoritmo.
- Más facilidad para reutilización de código.

Tipos de Módulos

- Procedimientos.
- Funciones.

Función

- Toma uno o más valores (argumentos o parámetros actuales) y devuelve un resultado en el nombre de la función.
- Se invoca utilizando su nombre seguido de los parámetros actuales o reales entre paréntesis.
- Parámetros formales: nombre que reciben los parámetros en la propia definición o declaración de la función.
- Parámetros actuales: valores reales que toman los parámetros en las llamadas a la función.

Una función debe devolver un valor.

Función

- Consta de:
  - Cabecera: definición de la función y parámetros formales necesarios. Los parámetros formales pueden ir acompañados de tres posibles modificadores: E (entrada), S (salida), E/S (entrada/salida).
  - Cuerpo: declaración de parámetros formales e instrucciones. Debe incluir instrucción para devolver un determinado valor al algoritmo principal.
- Tipos:
  - Internas (intrínsecas): funciones estándar, que realizan cálculos habituales, ya han sido programadas y pueden ser utilizadas por cualquier persona usuaria.
  - Externas (extrínsecas): definidas por cada usuario/a.

Función

- Construcción:

```
<tipo_dato> FUNCIÓN <nombre_función>(<parámetros>)
INICIO
    INSTRUCCIONES
    DEVOLVER (<expresión>)
FIN_FUNCIÓN
```

- Ejemplo pseudocódigo:

```
Entero FUNCIÓN Absoluto (E Entero: x)
INICIO
    SI x < 0
        DEVOLVER (-1 * x)
    SI_NO
        DEVOLVER (x)
FIN_FUNCIÓN
```

Procedimiento

- Subalgoritmo que realiza una tarea específica y puede definirse con 0, 1 o N parámetros.
- Útil para agrupar secuencias de sentencias que deben ser realizadas juntas.
- La estructura y lista de parámetros formales siguen las mismas reglas que en el caso de las funciones.
- Se invoca utilizando su nombre seguido de los parámetros actuales o reales entre paréntesis.

Un procedimiento no devuelve ningún valor.

Procedimiento

- Construcción:

```
PROCEDIMIENTO <nombre_procedimiento>(<parámetros>)
INICIO
   INSTRUCCIONES
FIN_PROCEDIMIENTO
```

- Ejemplo pseudocódigo:

```
PROCEDIMIENTO Area (E real: l1, l2; S real: a)
INICIO
   a = l1 * l2
   IMPRIMIR (a)
FIN_PROCEDIMIENTO
```

Parámetros

- En llamadas a funciones o procedimientos se establece correspondencia entre parámetros formales y reales.
- Formas de correspondencia:
  - Posicional: se emparejan los parámetros reales y formales según su posición en la lista: mismo número de parámetros en las dos listas.
    - En definición hay que indicar tipo de parámetro formal.
  - Por nombre explícito: se indica de forma explícita la correspondencia entre los parámetros actuales y los formales.

Tipos de Parámetros

- Tres tipos :
  - Entrada (E): transmiten información del programa que invoca al subprograma.
  - Salida (S): devuelven resultados.
  - Entrada/Salida (E/S): mandan valores al subprograma o devuelven resultados.
- Función puede devolver valores al programa principal:
  - Como valor de la función.
  - A través de parámetros.
- Procedimiento solo puede devolver resultados a través de parámetros.

Paso de Parámetros

Paso por valor

- Los parámetros formales reciben una copia del valor de los parámetros actuales.
- Los parámetros actuales no se ven afectados por las modificaciones que se produzcan en sus copias dentro de cada función.

Paso por referencia

- Los parámetros formales no reciben una copia de los actuales: se pasa directamente dirección de memoria.
- Cualquier modificación de estos parámetros dentro de la función modifica el valor de los parámetros actuales.

En C, para el paso por referencia: &

Variables Globales y Locales

Variables Globales

- Son conocidas en el ámbito del programa completo.
- Se declaran en el programa principal.

Variables Locales

- Se declaran dentro de una función o procedimiento, y solo se conocen en este ámbito.
- Puede haber otras con el mismo nombre en el programa principal.

Ejemplo

```
VARIABLES
    contador, limite : Entero
ALGORITMO calcular_pares
    ……
    contar_pares(limite)
    ……
PROCEDIMIENTO contar_pares (E Entero : limite)
   contador : Entero
    ……
```

Referencias

Joyanes Aguilar, L.; Rodríguez Baena, L.; Fernández Azuela, M. (2008). Fundamentos de programación. McGraw-Hill.

García-Bermejo Giner, J.R. (2008). Programación estructurada en C, Pearson Prentice Hall.

Joyanes Aguilar, L.; Zahonero Martínez, I. (2007). Programación en C : metodología, algoritmos y estructura de datos.