Taller de Programación

06 - Subalgoritmos

Mgs. Lic. Marcos Prunello

2018

Subalgortimos

- Principio fundamental en la resolución de un problema: descomponerlo en partes más pequeñas, que puedan ser más fáciles de afrontar.
- Los algoritmos pueden descomponerse en **subalgoritmos** que den solución a un aspecto del problema, de menor extensión: **descomposición algorítmica**.
- Cada subalgoritmo debe ser independiente de los demás y a su vez podría seguir descomponiéndose en partes más sencillas (**refinamiento sucesivo**).
- Un subalgoritmo se escribe una vez y luego es utilizado por todos aquellos algoritmos que lo necesiten.

Ejemplo: algoritmo para el cálculo de un número combinatorio entre n y k = n! / ((n-k)! k!)

```
ALGORITMO: Ej 1. Cálculo de números combinatorios
COMENZAR
    VARIABLE numerica n, k, fact1, fact2, fact3, comb
    LEER n, k \\ Asumimos que n y k cumplen con los requisitos

    \\ Calcular el factorial de n
    fact1 <- 1
    PARA i DESDE 1 HASTA n HACER
        fact1 <- fact1 * i
    FIN PARA

    \\ Calcular el factorial de n-k
    fact2 <- 1
    PARA i DESDE 1 HASTA n - k HACER
        fact2 <- fact2 * i
    FIN PARA

    \\ Calcular el factorial de k
    fact3 <- 1
    PARA i DESDE 1 HASTA k HACER
        fact3 <- fact3 * i
    FIN PARA

    \\ Calcular el nro combinatorio
    comb <- fact1 / (fact2 * fact3)

    ESCRIBIR "El nro combinatorio de " n " tomado de a " k " es " comb
FIN
```

Ejemplo: algoritmo para el cálculo de un número combinatorio.

- El cálculo del factorial requiere siempre la misma estructura y se repite tres veces.
- Esto puede plantearse por separado, dando lugar a un subalgoritmo.
- El algoritmo quedaría mejor expresado así:

```
ALGORITMO: Ej 2. Cálculo de números combinatorios
COMENZAR
    VARIABLE numérica n, k, comb
    LEER n, k
    comb <- factorial(n) / (factorial(n - k) * factorial(k))
    ESCRIBIR "El nro combinatorio de " n " tomado de a " k
             " es " comb
FIN
```

Ejemplo: algoritmo para el cálculo de un número combinatorio.

- Se hizo uso de un subalgoritmo llamado `factorial` que toma entre paréntesis un valor para el que procede a calcular y devolver el factorial.
- El mismo debe ser definido aparte dicho subalgoritmo, como se muestra a continuación:

```
FUNCIÓN factorial(n: numérico): numérico
    VARIABLE numérica fact
    fact <- 1
    PARA i DESDE 1 HASTA n HACER
        fact <- fact * i
    FIN PARA
    DEVOLVER fact
FIN FUNCIÓN
```

Más adelante: detalles de esta definición y por qué decimos que es una *función*

Ventajas del uso de subalgoritmos

- **En algoritmos complejos**: si el algoritmo, y luego el programa, es muy largo y se escribe todo seguido resulta complicado de entender, se pierde la visión de su estructura global. Aislando ciertas partes como subalgoritmos separados se reduce la complejidad.
- **Cuando se repiten operaciones análogas**: si hay una tarea que se repite varias veces, podemos definirla como un subalgoritmo por separado. El código se escribe sólo una vez aunque se use en muchos puntos del programa.

Funciones

- Los subalgoritmos se clasifican en **funciones** y **procedimientos**.
- Una **función** es un subalgoritmo que devuelve un único resultado.
- En programación, la noción de función se asemeja a la idea matemática de función de una o más variables. Por ejemplo, podemos pensar en la función $f(x, y) = x^2 + 3y$.
- En pseudocódigo, el subalgoritmo que se encargaría de implementarla es:

```
FUNCIÓN f(x: numérico, y: numérico): numérico
COMENZAR
    DEVOLVER x ** 2 + 3 * y
FIN FUNCIÓN
```

Funciones

En el ejemplo anterior:

- La función es de tipo numérico, porque devuelve un valor numérico (podría ser caracter o lógico).
- x e y son los **parámetros formales** o **ficticios** y son los que permiten expresar la "ley" o "forma" de la función.
- Se aclara en el encabezado que estos parámetros son de tipo numérico.
- Los valores en los cuales se quiere evaluar la función se llaman **parámetros actuales** o **reales**.
- Ej: para calcular $f(4, 5)$, los valores 4 y 5 son los parámetros actuales y se establece una correspondencia entre el parámetro formal x y el real 4, así como entre la y y el 5. La función devuelve el valor 31.
- A los parámetros también se les dice **argumentos**.

Funciones

Otra forma de escribir la función anterior:

```
FUNCIÓN f(x: numérico, y: numérico): numérico
COMENZAR
    VARIABLE numérica rtdo
    rtdo <- x ** 2 + 3 * y
    DEVOLVER rtdo
FIN FUNCIÓN
```

- Observar que podemos crear nuevas variables dentro de la función.

Funciones

De manera general, la definición de una función es:

```
FUNCIÓN nombre(lista de parámetros formales): tipo de rtdo
COMENZAR
    Declaración de variables
    Acciones
    DEVOLVER valor
FIN FUNCIÓN
```

Funciones: la palabra clave DEVOLVER

- Pprovoca la inmediata finalización de la ejecución de la función e indica cuál es el resultado de la misma
- Se puede insertar en cualquier punto de la parte ejecutable de la función
- Se puede utilizar más de una sentencia DEVOLVER, aunque sólo una llegue a ejecutarse:

```
FUNCIÓN maximo(num1: numérico, num2: numérico): numérico
COMENZAR
    SI num1 >= num2
        ENTONCES
            DEVOLVER num1
        SI NO
            DEVOLVER num2
    FIN SI
FIN FUNCIÓN
```

Funciones: invocación

- Para usar una función en un algoritmo, se la invoca escribiendo su nombre seguida por los valores actuales entre paréntesis, separados por coma.
- Los valores suministrados para los argumentos reales deben corresponder en cantidad, tipo y orden con los argumentos formales de la definición de la función. Por ejemplo:

```
ALGORITMO: Hallar el máximo entre dos valores
COMENZAR
    VARIABLE numérica x, y
    LEER x, y
    ESCRIBIR "El máximo es " maximo(x, y)
FIN
```

Procedimiento

- Es un subalgoritmo que agrupa una acción o conjunto de acciones, dándoles un nombre por el que se las puede identificar posteriormente.
- A diferencia de la *función*, no tiene como objetivo general devolver un valor, pudiendo devolver ninguno, uno o varios (por eso no declaramos de qué *tipo* es).
- El objetivo principal es ayudar en la modularidad del programa y evitar la repetición de acciones.
- Puede o no incluir parámetros
- Para invocarlo desde el algoritmo principal, se escribe su nombre seguido por los valores de los argumentos actuales, entre paréntesis, separados por comas (si incluye parámetros, respetando cantidad, tipo y orden).

Procedimiento. Ejemplo 1.

- Definimos un procedimiento que se encargue de escribir un título con recuadro y otro para escribir una línea:

```
PROCEDIMIENTO colocarTitulo(titulo: caracter)
    ESCRIBIR "================================="
    ESCRIBIR titulo
    ESCRIBIR "================================="
FIN PROCEDIMIENTO

PROCEDIMIENTO colocarLinea()
    ESCRIBIR "_________________________________"
FIN PROCEDIMIENTO
```

Procedimiento. Ejemplo 1.

- Entonces, el algoritmo principal podría incluir:

```
ALGORITMO: Ej 4. Procedimientos con y sin argumentos
COMENZAR
    ...
    colocarTitulo("Primer grupo de resultados")
    ESCRIBIR 1
    colocarLinea()
    ESCRIBIR 2
    colocarLinea()
    ESCRIBIR 3

    colocarTitulo("Segundo grupo de resultados")
    ESCRIBIR 4
    colocarLinea()
    ESCRIBIR 5
    colocarLinea()
    ESCRIBIR 6
FIN
```

Procedimiento. Ejemplo 1.

- Como resultado la salida mostrará:

```
==================================
Primer grupo de resultados
==================================
1

----------------------------------
2

----------------------------------
3


==================================
Segundo grupo de resultados
==================================
4

----------------------------------
5

----------------------------------
6
```

Procedimiento. Ejemplo 2.

```
ALGORITMO: Ejemplo 5
COMENZAR
    VARIABLE numérica a, b, c, d
    a <- 5
    b <- 2
    proc1(a, b, c, d)
    ESCRIBIR a b c d
FIN

PROCEDIMIENTO proc1(n1: numérico, n2: numérico,
                    n3: numérico, n4: numérico)
    n3 <- n1 + n2
    n4 <- n2 - 1
FIN PROCEDIMIENTO
```

Procedimiento. Ejemplo 2.

- En el ejemplo 2 identificamos los argumentos reales `a` (con el valor 5), `b` (con el valor 2), `c` y `d` (sin valores asignados inicialmente).
- Cuando el procedimiento `proc1` es invocado, se establece una correspondencia con los argumentos formales `n1`, `n2`, `n3` y `n4`, respectivamente.
- `n1` toma el valor 5, `n2` toma el valor 2 y el procedimiento le asigna los valores 7 a `n3` y 1 a `n4`.
- Al finalizar, este procedimiento habrá dejado sin cambios a las variables `a` y `b`, mientras que le habrá asignado los valores 7 a `c` y 1 a `c`.
- Como resultado, el algoritmo escribe "5 2 7 1".

Procedimiento. Ejemplo 3.

```
ALGORITMO: Ejemplo 6
COMENZAR
    VARIABLE numérica a, b
    a <- 5
    b <- 2
    proc2(a, b)
    ESCRIBIR a b
FIN

PROCEDIMIENTO proc2(n1: numérico, n2: numérico)
    n1 <- n1 + n2
    n2 <- n2 - 1
FIN PROCEDIMIENTO
```

Procedimiento. Ejemplo 3.

- En el ejemplo 3, el procedimiento `proc2` modifica las variables que actúan como argumentos reales.
- Al ser invocado, se establece una correspondencia entre los argumentos reales `a` (con el valor 5) y `b` (con el valor 2), y los argumentos formales `n1` y `n2`, respectivamente.
- La primera acción del procedimiento le asigna el valor 7 a `n1` y 1 a `n2`.
- Al finalizar `a` vale 7 y `b` vale 1 y el algoritmo escribe "7 1".

Pasaje de argumentos

- Los algoritmos y subalgoritmos comunican información entre sí a través de los parámetros o argumentos y existen distintas formas de realizar esta comunicación.
- Veremos el pasaje de argumentos **por valor** y **por referencia**.

Pasaje de argumentos: por valor

- Los argumentos representan valores que se transmiten **desde** el algoritmo **hacia** el subalgoritmo.
- Los objetos del algoritmo provistos como argumentos en la llamada al subalgoritmo no serán modificados por la ejecución del mismo.
- Este sistema funciona de la siguiente forma:
  1. Se evalúan los argumentos reales usados en la llamada.
  2. Los valores obtenidos se copian en los argumentos formales dentro del subalgoritmo.
  3. Los argumentos formales se usan como variables dentro del subalgoritmo. Aunque los mismos sean modificados (por ejemplo, se les asignen otros valores), no se modifican los argumentos reales en el algoritmo, sólo sus copias dentro del subalgoritmo.

Pasaje de argumentos: por referencia

- Cuando un argumento es pasado por referencia, el subalgoritmo puede modificar las variables del algoritmo principal.
- De esta manera se puede producir ("devolver") más de un resultado.
- Si un parámetro se pasa por referencia, esta variable será empleada en el subalgoritmo como si fuera suya, es decir, las modificaciones que sufra dentro del algoritmo la modificarán permanentemente.
- Este sistema funciona de la siguiente forma:
  1. Se seleccionan las variables usadas como argumentos reales.
  2. Se asocia cada variable con el argumento formal correspondiente.
  3. Los cambios que experimenten los argumentos formales se reflejan también en los argumentos reales de origen.

Pasaje de argumentos: ejemplo 1

Algunos lenguajes de programación permiten que el programador elija el modo en el que se realiza el pasaje. En el siguiente ejemplo veremos la diferencia entre ambos modos.

```
ALGORITMO: Ejemplo 7
COMENZAR
    VARIABLE numérica a, b, c
    a <- 3
    b <- 5
    c <- fun(a, b - a)
    ESCRIBIR a b c
FIN

FUNCIÓN fun(x: numérico, y: numérico): numérico
    x <- x + 1
    DEVOLVER x + y
FIN FUNCIÓN
```

Pasaje de argumentos: ejemplo 1, por valor

- No se modifican las variables en el algoritmo principal, la función sólo tiene una "copia" de esos valores.
- Los pasos que se ejecutan son:
  - `a = 3, b = 5`
  - Al invocar la función: `x = 3, y = 5 - 3 = 2`
  - Primera línea de la función: `x = 3 + 1 = 4`
  - La función devuelve el valor `x + y = 4 + 2 = 6`
  - De regreso en el algoritmo principal: `c` recibe el valor `6`
  - Se escribe: `3 5 6`

Pasaje de argumentos: ejemplo 1, por referencia

- Cualquier modificación realizada dentro de la función es automáticamente realizada también a las respectivas variables en el algoritmo principal.
- Los pasos que se ejecutan son:
  - `a = 3, b = 5`
  - Al invocar la función: `x = 3, y = 5 - 3 = 2`
  - Primera línea de la función: `x = 3 + 1 = 4`. El parámetro actual asociado con `x`, `a`, sufre el mismo cambio y recibe el valor `4` (`a = 4`).
  - La función devuelve el valor `x + y = 4 + 2 = 6`
  - De regreso en el algoritmo principal: `c` recibe el valor `6`
  - Se escribe: `4 5 6`

Pasaje de argumentos: ejemplo 2

- Analicemos ahora el tipo de pasaje en el contexto de un procedimiento

```
ALGORITMO: Ejemplo 8
COMENZAR
    VARIABLE numérica a, b
    a <- 8
    b <- 4
    miProc(a, b)
    ESCRIBIR a b
FIN

PROCEDIMIENTO miProc(x: numérico, y: numérico)
    x <- x * 2
    y <- x - y
FIN PROCEDIMIENTO
```

Pasaje de argumentos: ejemplo 2, por referencia

- Los pasos que se ejecutan son:
  - `a = 8, b = 4`
  - Al invocar la función: `x = 8, y = 4`
  - Primera línea de la función: `x = 8 * 2 = 16`. Lo mismo sucede con el parámetro actual `a`: `a = 16`.
  - Segunda línea de la función: `y = 16 - 4 = 12`. Lo mismo sucede con el parámetro actual `b`: `b = 12`.
  - Al regresar al algoritmo principal, la sentencia `ESCRIBIR` produce: `16 12`.

Pasaje de argumentos: ejemplo 2, por valor

- Si el pasaje hubiese sido por valor, `a` y `b` no hubiesen cambiado y la sentencia `ESCRIBIR` mostraría `8, 4`.
- Como en un procedimiento los resultados regresan en los mismos parámetros, no pueden ser todos pasados por valor, porque en ese caso el procedimiento nunca realizaría ninguna acción.

Pasaje de argumentos: ejemplo 2, `x`por valor, `y` por referencia

- Los pasos que se ejecutan son:
  - `a = 8, b = 4`
  - Al invocar la función: `x = 8, y = 4`
  - Primera línea de la función: `x = 8 * 2 = 16`.
  - Segunda línea de la función: `y = 16 - 4 = 12`. Lo mismo sucede con el parámetro actual `b`: `b = 12`.
  - Al regresar al algoritmo principal, la sentencia `ESCRIBIR` produce: `8 12`.

Variables locales y globales. Transparencia referencial

- Leer en la guía.