Lenguaje de Diseño en PSeInt
Subalgoritmos

RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

Ingeniería en Informática
Ingeniería en Computación
Profesorado en Computación

UNIVERSIDAD NACIONAL DE SAN LUIS
DEPARTAMENTO DE INFORMÁTICA
AÑO 2024

Índice

Subalgoritmos

1. Introducción

Dado un problema relativamente complejo, la manera más conveniente de resolverlo es dividirlo en tareas o módulos de menor complejidad. A estos módulos, en lenguaje de diseño se los denomina **subalgoritmos, subprogramas o subprocesos**.

Un subalgoritmo tiene que ser **lógicamente coherente y autocontenido** y formar parte de un algoritmo de mayor envergadura. Por **lógicamente coherente** se quiere significar que contiene un conjunto determinado de acciones que conducen a la solución de una tarea específica, relativamente simple; mientras que **autocontenido** significa que es independiente de cualquier otro problema y de sus soluciones. Los subalgoritmos son una facilidad del lenguaje de diseño (y de los lenguajes de programación) que permiten que determinados conjuntos de acciones puedan ser provistos en una forma modular. La **característica** de este conjunto de acciones es que realizan tareas comunes que pueden utilizarse para resolver distintos tipos de problemas o bien, varias veces dentro de un mismo problema, pero con distintos datos.

Estos subalgoritmos se escriben una vez y, luego, son usados por todos aquellos que requieran de ellos. Por ejemplo: ordenar datos en orden ascendente/descendente, como así también las funciones primitivas propias del lenguaje como por ejemplo *valor absoluto* (abs), *raíz cuadrada* (rc), *número aleatorio* (azar), entre otros.

Lo expresado puede verse gráficamente en la Figura 1:

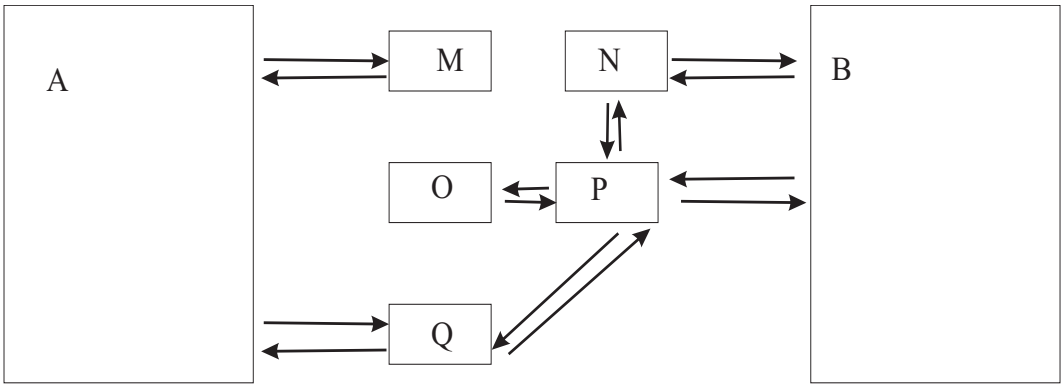

Figura 1

En esta figura los subalgoritmos M, N, P, O y Q han sido escritos una vez y son usados indistintamente por

los algoritmos A y B, o entre ellos.

2. Definición de Subalgoritmos en PSeInt

En PSeInt se distinguen dos clases de subprocesos: **funciones** que devuelven un resultado y **procedimientos** que realizan un proceso específico y no necesariamente requiere de datos para llevarla a cabo y/o devuelve resultados.
Los subalgoritmos se definen colocando la palabra reservada **SUBALGORITMO**.

SINTÁXIS DE DECLARACIÓN DE FUNCIONES

Los subalgoritmos que van a ser utilizados como **funciones** (es decir, devuelven un resultado) se definen colocando la palabra reservada **SubAlgoritmo** seguida de la variable de retorno, la especificación de su nombre y, opcionalmente, los argumentos o parámetros (los términos *argumento* y *parámetro* son sinónimos). En el cuerpo de la función se realiza la declaración de sus propias variables (*variables locales*) y la secuencia de acciones que hacen a la logística propia del mismo.
Es obligatorio **declarar** la variable de retorno y **asignarle** algún valor y solamente se puede definir una única variable de retorno, es decir, la función devuelve un único resultado.

Sintaxis de definición de una Función

SubAlgoritmo <**Nombre de la variable de retorno**> $\longleftarrow$ $Nombre$ (<Nombre del parámetro 1>, <Nombre del parámetro 2>, . . .)
Declaracion de variables

Secuencia de acciones
FinSubAlgoritmo

El nombre de un subalgoritmo se construye siguiendo las mismas restricciones que para las variables y los arreglos.

Los parámetros en un subalgoritmo son canales de comunicaión entre quien invoca y el módulo invocado. Los parámetros que aparecen en la definición de un subalgoritmo se conocen con el nombre de **parámetros formales** y **se utilizan sólo** en las **acciones** que conforman el **cuerpo** del subalgoritmo, al igual que las variables locales que allí se definan. Estos parámetros permiten ingresar datos al subproceso u obtener resultados desde el mismo y que serán comunicados al módulo que lo invocó. Para utilizar el subalgoritmo es

necesario invocarlo a través del nombre y la lista de argumentos necesarios para que el mismo funcione. Los parámetros utilizados en el momento de la invocación reciben el nombre de **parámetros actuales o reales**. El número de parámetros actuales debe ser igual al número de parámetros formales.

Pueden definirse dos clases diferentes de parámetros formales:

- **Por valor** son sólo **parámetros de entrada**. Implica que el argumento formal recibe una copia del contenido de la variable (o el resultado de una expresión) que se utilizó al momento de realizar la llamada o **invocación** del subprograma. Las modificaciones que se realicen a esos valores en el cuerpo del módulo no se reflejaran fuera del mismo. Si el parámetro formal es una variable simple y no se define el tipo de pasaje, se asume que es por valor.
- **Por referencia:** son **parámetros de entrada y de salida**, es decir el subalgoritmo puede devolver resultados de la ejecución de sus acciones a quien lo invocó. Tanto el parámetro formal como la variable utilizada al momento de invocar al subalgoritmo (parámetro actual) comparten la misma posición de memoria. Es por eso que, cualquier cambio que se realice al parámetro formal en el cuerpo del módulo, se refleja directamente en el respectivo parámetro actual. Si el parámetro formal es un arreglo y no se define el tipo de pasaje, se asume que es por referencia.

Ejemplo de Función

SubAlgoritmo $EsnoES \longleftarrow$ Es-raiz-cuadrada (xx, yy)

Definir EsnoEs como logico

EsnoEs $\leftarrow$ FALSO

Si xx = rc(yy) entonces

EsnoEs $\leftarrow$ VERDADERO

FinSi

FinSubAlgoritmo

Donde:

- ***EsnoEs*** es la **variable de retorno** que recibirá el resultado de ejecutar la función.
- ***Es-raiz-cuadrada*** es el **nombre** de la función.

- ***xx e yy*** son los **parámetros formales** y el pasaje es por **valor**

Algoritmo Raiz-Cuadrada
Definir A,B como enteros
Definir Respuesta como logico
Leer "Ingresar valor"
Leer A
Leer "Ingresar valor"
Leer B
// Invocacion de la Funcion Es-raiz-cuadrada
Respuesta ⟵ **Es-raiz-cuadrada(A, B)**

Escribir Respuesta

FinAlgoritmo

Donde:

- ***Respuesta*** es la **variable** que recibirá el resultado de invocar a la función.

Los subalgoritmos que van a ser utilizados como **procedimientos** NO necesitan de una variable de retorno. Se definen colocando la palabra reservada **SubAlgoritmo** seguida de la especificación de su nombre y, opcionalmente, los parámetros que sean necesarios. En el cuerpo del subproceso se realiza la declaración de sus propias variables (*variables locales*) y la secuencia de acciones que hacen a la logística propia del mismo.
No se **declara** variable de retorno y en los casos que sea necesario que el procedimiento retorne resultados a quien lo invocó, **deberá** hacerlo a través del pasaje de parámetros por referencia. De esta manera podrá retornar más de un resultado.

Sintaxis de definición de un Procedimiento

SubAlgoritmo $Nombre$ (<Nombre del parámetro 1>, <Nombre del parámetro 2>, ...)

Declaracion de variables
Secuencia de acciones

FinSubAlgoritmo

Ejemplo de Procedimiento

SubAlgoritmo Factorial (**N**, **FAC** por referencia)
Definir I como entero
FAC $\longleftarrow$ 1
Para $I \longleftarrow$ 1 **Hasta** N **Con Paso** 1 **Hacer**
FAC $\longleftarrow$ FAC$*I$
FinPara
FinAlgoritmos

Algoritmo Factorial-de-un-numero
// Este programa calcula el factorial del valor N, N^2 y N^3
Definir N,RESUL como enteros
Escribir "Ingrese el valor al que le desea calcular el factorial"
Leer N
// Invocacion del Procedimiento Factorial para N
Factorial (N, RESUL)
Escribir "El factorial de ", N "es", $RESUL$
// Invocacion del Procedimiento Factorial para N^2
Factorial (N$*N$, RESUL)
Escribir "El resultado es", $RESUL$
// Invocacion del Procedimiento Factorial para N^3
Factorial (N^ 3, RESUL)
Escribir "El resultado es", $RESUL$

FinAlgoritmo

En el ejemplo del subalgoritmo FACTORIAL, el parámetro N es un parámetro por valor (no se ha especificado ningún tipo de pasaje de parámetro), que le provee al subalgoritmo el valor del número para el cual se debe calcular el factorial, mientras que el parámetro FAC es un parámetro de salida (por referencia) y es el argumento que devuelve al módulo que lo invocó el resultado del cálculo del factorial.

Notar que en la segunda y tercera invocación de FACTORIAL se ha pasado como primer argumento una expresión. Esta pasaje es válido debido a que el primer parámetro ha sido definido por valor.

Las variables que se declaran dentro de un subalgoritmo sólo pertencen al ambiente del subalgoritmo donde están definidas y se conocen con el nombre de **variables locales**. Tienen el mismo ámbito de uso que los parámetros formales del subalgoritmo. En este ejemplo es el caso de la variable I. El **ámbito** se define entre las palabras reservadas **SubAlgoritmo** y **FinSubAlgoritmo** y la secuencia de acciones comprendidas entre dichas sentencias forman parte del cuerpo del subalgoritmo.

3. Definición, Invocación y Retorno de Subalgoritmos

Todos los subalgoritmos o subprogramas deben definirse antes del algoritmo principal y antes de ser invocados. Se **invoca** al subalgoritmo através de su nombre, seguido de la lista de argumentos actuales, los cuales deben coincidir en
Para que las acciones descriptas en el subalgoritmo sean ejecutadas, es necesario que el mismo **sea invocado desde el algoritmo principal ó desde otro subalgoritmo**.

Es importante aclarar que si el subalgoritmo fue definido con una determinada cantidad de parámetros formales, debe ser invocado con esa misma cantidad de parámetros actuales.

Cada vez que un subalgoritmo es invocado se establece, automáticamente, una **correspondencia** entre los parámetros formales y los actuales. Esta correspondencia está definida **por la posición que los parámetros ocupan dentro de la lista de parámetros**, es decir, el primer parámetro actual se corresponde con el primer parámetro formal; el segundo parámetro actual con el segundo parámetro formal y, así siguiendo, hasta completar todos los parámetros.

Cuando se invoca a un subalgoritmo, **la ejecución del módulo que invoca se suspende y el control de la ejecución se transfiere a la primera acción del cuerpo del subalgoritmo invocado**. Luego de la ejecución de las acciones que describen la lógica del cuerpo del subalgoritmo, el **control de la ejecución** retorna a la sentencia siguiente a la invocación del subalgoritmo en el módulo invocante.

Para los **parámetros formales** que fueron definidos:

- **por valor:** los **parámetros actuales** pueden ser constantes, variables (definidas en el ambiente del módulo invocante), expresiones o valores de funciones de PSeInt.
- **por referencia:** los **parámetros actuales** deben ser variables simples o arreglos definidos en el ambiente del módulo invocante, pues allí el subalgoritmo devuelve sus resultados (tanto el parámetro formal como el actual comparten las posiciones de memoria asignadas a esas variables).

4. ANEXO: Ejemplos en PSeInt

4.1. Ejemplo 1:

¿Qué resuelve el siguiente programa?. Redacte el enunciado.

```
// función que recibe un argumento por valor, y devuelve su doble
SubAlgoritmo res ← CalcularDoble(num)
    Definir res como real
    res ← num*2 // retorna el doble
FinSubAlgoritmo

// procedimiento que recibe un argumento por referencia, y lo modifica
SubAlgoritmo Triplicar(num por referencia)
    num ← num*3 // modifica la variable duplicando su valor
FinSubAlgoritmo
// procedimiento que recibe un argumento por referencia y otro por valor,
//y modifica al argumento por referencia
SubAlgoritmo es_multiplo(esnoes por referencia, num1 por valor, num2 por valor)
    Definir Resto como entero
    Resto ← num1%num2 // modifica la variable duplicando su valor
    esnoes← FALSO
    Si (Resto = 0) Entonces
        esnoes← VERDADERO
    FinSi
FinSubAlgoritmo



// proceso principal, que invoca a las funciones antes declaradas

Algoritmo PruebaFunciones
    Definir x Como Real
    Definir esmultiplo como logico
    Definir N1, N2 como enteros
    Escribir "Ingrese un valor numérico para x:"
    Leer x
    Escribir "Llamada a la función CalcularDoble (pasaje por valor)"
    Escribir "El doble de ",x," es ", CalcularDoble(x)
    Escribir "El valor original de x es ",x
    Escribir "Llamada al procedimiento Triplicar (pasaje por referencia)"
    Triplicar(x)
    Escribir "El nuevo valor de x es ", x
    Escribir "Llamada al procedimiento esnoes (pasaje por referencia y pasaje por valor)"
    Escribir "Ingrese un valor"
    Leer N1
    Escribir "Ingrese un valor"
    Leer N2
    Escribir "Llamada al procedimiento es_multiplo (pasaje por referencia y por valor)"
    esmultiplo← FALSO
    es_multiplo(esmultiplo, N1, N2)
    Escribir "N1 es multiplo de N2? ", esmultiplo

FinAlgoritmo
```

4.2. Ejemplo 2:

El dueño de un gimnasio quiere hacer un estudio sobre los planes de sus socios. Por cada cliente se tienen los siguientes datos: el tipo de actividad del plan contratado (aparatos, fitness o natación) y la cantidad de clases semanales de las que dispone (2, 4 o libre). Se pide: Diseñar un procedimiento que permita al usuario consultar por la cantidad de socios que han contratado un tipo de actividad en particular especificando, además la cantidad de clases semanales que disponen.

```
SubAlgoritmo ingreso(socios2,li2, ls2 por referencia)
    definir i Como Entero
    Escribir "Ingrese la cantidad de socios que quiere cargar"
    Leer ls2
    Mientras ls2 < 1 ∨ ls2 >100 Hacer
        Escribir "Ingrese la cantidad de socios que quiere cargar, entre 1 y 100"
        Leer ls2
    FinMientras
    Para i ← li2 Hasta ls2*2 con paso 2 Hacer
        Escribir "Ingrese el tipo de actividad: 1.Aparatos, 2.Fitness, 3.Natación"
        Leer socios2[i]
        Mientras socios2[i] ≠ 1 ∧ socios2[i] ≠ 2 ∧ socios2[i] ≠ 3 Hacer
            Escribir "Ingrese el tipo de actividad correctamente: 1.Aparatos, 2.Fitness, 3.Natación"
            Leer socios2[i]
        FinMientras
        Escribir "Ingrese la cantidad de clases: 1.2 veces por semana, 2.4 veces por semana, 3.Libre"
        Leer socios2[i+1]
        Mientras socios2[i+1] ≠ 1 ∧ socios2[i+1] ≠ 2 ∧ socios2[i+1] ≠ 3 Hacer
            Escribir "Ingrese la cantidad de clases correctamente: 1.2 veces por semana, 2.4 veces por semana, 3.Libre"
            Leer socios2[i+1]
        FinMientras
    FinPara
FinSubAlgoritmo
```

```
SubAlgoritmo consultar(socios2,li2, ls2, tipo2, clases2, cantidad2 por referencia )
    definir i Como Entero
    cantidad2 ← 0
    Para i ← li2 Hasta ls2*2 con paso 2 Hacer
        Si socios2[i] == tipo2 ∧ socios2[i+1] == clases2  Entonces
            cantidad2 ←  cantidad2 +1
        FinSi
    FinPara
FinSubAlgoritmo
```

```
Algoritmo P7E5
    Definir tipo, clases, res,li,ls Como Entero
    Definir socios Como Entero
    Dimension socios[200]
    li← 1
    ingreso(socios,li,ls)
    Escribir "Ingrese el tipo de actividad a consultar : 1.Aparatos, 2.Fitness, 3.Natación"
    Leer tipo
    Mientras tipo ≠ 1 ∧ tipo ≠ 2 ∧ tipo ≠ 3 Hacer
        Escribir "Ingrese el tipo de actividad a consultar correctamente: 1.Aparatos, 2.Fitness, 3.Natación"
        Leer tipo
    FinMientras
    Escribir "Ingrese la cantidad de clases a consultar: 1.2 veces por semana, 2.4 veces por semana, 3.Libre"
    Leer clases
    Mientras clases ≠ 1 ∧ clases ≠ 2 ∧ clases ≠ 3 Hacer
        Escribir "Reingrese la cantidad de clases a consultar: 1.2 veces por semana, 2.4 veces por semana, 3.Libre"
        Leer clases
    FinMientras
    consultar(socios, li, ls, tipo, clases, res)
Escribir "La cantidad de clientes con el tipo de actividad", tipo, "y la cantidad de clases", clases, "es: ", res
FinAlgoritmo
```