Programación estructurada

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de ordenador o algoritmo, utilizando únicamente subrutinas (funciones o procedimientos) y tres estructuras: secuencia, alternativas y repetitivas.

La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún algoritmo (divide y vencerás). Además el uso de subrutinas nos proporciona la reutilización del código y no tener repetido instrucciones que realizan la misma tarea.

Subrutinas en pseudocódigo

Tenemos dos clases de subrutinas:

- **Funciones**: Subrutina que devuelve un valor.
- **Procedimientos**: Subrutina que no devuelve ningún resultado.

Para definir una función o procedimiento:

```
Funcion [variable_de_retorno <-] nombre_de_la_funcion ( argumento_1,
argumento_2, ... )
      acción 1;
      acción 1;
            .
            .
            .
      acción n;
 FinFuncion
```

- Comienza con la palabra clave `Funcion` (alternativamente puede utilizar `SubProceso` o `SubAlgoritmo`, son sinónimos) seguida de la variable de retorno, el signo de asignación, el nombre del subproceso, y finalmente, la lista de argumentos entre paréntesis.
- Si el subproceso no recibe ningún valor pueden colocarse los paréntesis vacíos u omitirse, finalizando la primer línea con el nombre del subproceso.
- Si estamos trabajando con un procedimiento se coloca directamente el nombre y los argumentos a continuación de la palabra clave `Funcion`.

Ejemplo de función

Vamos a crear una función que calcule el valor máximo de dos números:

```
Funcion max <- CalcularMaximo(num1,num2)
        Definir max Como Entero;
        Si num1>num2 Entonces
                max <- num1;
        SiNo
                max <- num2;
        FinSi
FinFuncion
```

Como vemos la variable que se devuelve `max` hay que definirla en la función.

Para utilizar dicha función en el programa principal:

```
Proceso Maximo
        Definir numero1,numero2,num_maximo Como Entero;
        Escribir "Dime el número1:";
        Leer numero1;
        Escribir "Dime el número2:";
        Leer numero2;
        num_maximo <- CalcularMaximo(numero1,numero2);
        Escribir "El máximo es ",num_maximo;
FinProceso
```

Ejemplo de procedimiento

Vamos a escribir un procedimiento que recibe una cadena de caracteres y lo muestra en pantalla subrayado. No devuelve ningún valor.

```
Funcion Subrayar(cad)
        Definir i Como Entero;
        Escribir cad;
        Para i<-1 hasta Longitud(cad) hacer
                Escribir Sin Saltar "-";
        FinPara
FinFuncion

Proceso Titulos
        Definir titulo como cadena;
        titulo <- "Ejercicio 1";
        Subrayar(titulo);
        Escribir "";
FinProceso
```

Funciones y Procedimientos

Partimos del ejemplo anterior de función:

```
Funcion max <- CalcularMaximo(num1,num2)
        Definir max Como Entero;
        Si num1>num2 Entonces
                max <- num1;
        SiNo
                max <- num2;
        FinSi
FinFuncion

Proceso Maximo
        Definir numero1,numero2,num_maximo Como Entero;
        Escribir "Dime el número1:";
        Leer numero1;
        Escribir "Dime el número2:";
        Leer numero2;
        num_maximo <- CalcularMaximo(numero1,numero2);
        Escribir "El máximo es ",num_maximo;
FinProceso
```

Ámbito de variables

Las variables definidas en la función no existen en otras funciones o el programa principal. Igualmente las variables del programa principal no existen en la función.

Por ejemplo la variable `max` definida en la función no existe en el programa principalmente. Igualmente la variable `numero1` definida en el programa principal no existe en la función.

Parámetros formales y reales

- **Parámetros formales**: Son las variables que recibe la función, se crean al definir la función. Su contenido lo recibe al realizar la llamada a la función de los parámetro reales. Los parámetros formales son variables locales dentro de la función.
- **Parámetros reales**: Son la expresiones que se utilizan en la llamada de la función, sus valores se copiarán en los parámetros formales.

Paso de parámetro por valor o por referencia

- **Paso por valor**: El valor de los parámetros reales se copian en los parámetros formales, por lo tanto una modificación de algún parámetro formal no modifica el parámetro real.
- **Paso por referencia**: Cuando se pasa un parámetro por referencia implica que si modificamos el parámetro formal se modificará el parámetro real.

Pode defecto, los arreglos se pasan por referencia, las demás expresiones por valor. Si queremos indicar explícitamente como se pasan los parámetros podemos usar las palabras claves `Por Valor` o `Por Referencia`.

Ejemplos

Comprobamos que los parámetros pasados por valor no modifican los parámetros reales.

```
Funcion PasoPorValor(num)
        num <- num +1;
        Escribir num;
FinFuncion

Proceso Prueba
        Definir numero1 Como Entero;
        numero1<-5;
        PasoPorValor(numero1);
        Escribir numero1;
FinProceso
```

El resultado será 5 y 6. Hemos incrementado el valor del parámetro formal, pero no se ha modificado el real.

Veamos ahora el mismo programa pero pasando el parámetro por referencia.

```
Funcion PasoPorReferencia(num Por Referencia)
        num <- num +1;
        Escribir num;
FinFuncion
```

```
Proceso Prueba
        Definir numero1 Como Entero;
        numero1<-5;
        PasoPorReferencia(numero1);
        Escribir numero1;
FinProceso
```

El resultado será 6 y 6. Hemos modificado el parámetro formal y se modificado el real.

Llamada a la función

Para llamar a una función se debe utilizar su nombre y entre paréntesis los parámetros reales que se mandan. La llamada a una función se puede considerar una expresión cuyo valor y tipo es el retornado por la función. Evidentemente si estamos llamando un procedimiento, la llamada no tendrá ningún tipo.

Ejemplos de llamadas:

```
num1 <- CalcularMaximo(5,6)
Escribir CalcularMaximo(1,2)
...
```