Acciones Nominadas

1. Introducción

Cuando se diseña un algoritmo, muchas veces se necesita realizar varias veces un mismo proceso, por lo cual es necesario repetir varias veces las mismas instrucciones sobre diferentes datos para obtener un resultado. Esto hace que un algoritmo este formado por una gran cantidad de instrucciones, por lo cual se hace menos legible, más complejo y difícil de depurar. Además, dado que cada programador tiene su propia manera de pensar la solución para un problema, diseñar un algoritmo como un gran grupo de instrucciones ordenadas para lograr un propósito complejo hace que sea difícil la organización dentro de grupos de trabajo.

Para solucionar esta problemática, se considera mejor dividir el problema que se desea resolver en problemas más pequeños (subproblemas). Estos subproblemas, que deben responder a un propósito único y definido, pasan a ser pequeños algoritmos independientes llamados acciones nominadas, que pueden ser utilizados varias veces en diferentes partes de un algoritmo principal. Esta división lógica de un algoritmo en pequeñas partes independientes es el principio de la modularidad, ya que cada parte se puede ver como un módulo encargado de resolver un problema determinado.

2. Definición de Acción Nominada

Una acción nominada, también llamada subrutina, es un algoritmo, identificable mediante un nombre, que realiza determinadas tareas bien definidas por un grupo de instrucciones sobre un conjunto de datos. Las operaciones realizadas por una acción nominada son siempre las mismas, pero los datos sobre los que opera pueden variar cada vez que se utilice. Básicamente las acciones nominadas son miniprogramas dentro de un programa.

3. Características

- Permite lograr un objetivo.
- Tiene un nombre único que permite identificarla.
- Puede tener datos de entrada, los cuales llamamos parámetros o argumentos.
- Puede retornar un resultado, que debe concordar con el objetivo propuesto.
- Se usa como una instrucción.

4. Utilidad

- Permiten dividir un problema complejo en partes (subproblemas) más simples.
- Pueden ser reutilizadas.
- Facilitan la escritura y modificación de los algoritmos.
- Mejoran la legibilidad de los algoritmos.
- Permiten dividir el trabajo en equipos de programadores.

5. Tipos

Las acciones nominadas pueden ser divididas en dos grupos: Funciones y procedimientos.

5.1. Funciones

El concepto de función en programación se deriva del concepto de función matemática: "Una función es una relación que asocia con cada elemento de un conjunto llamado dominio, un único elemento de otro conjunto llamado codominio".

En programación, una función es una operación que puede recibir datos de entrada y retorna una salida. La definición de una función es la siguiente:

```
func nombre([F1; ...; Fn]): T
[var]
inicio
    // Instrucciones
    retornar(expr)
ffunc
```

Donde:

- *nombre*, es el identificador de la función.
- Cada F_i ($1 \leq i \leq n$), es un grupo de parámetros formales de la siguiente forma: {**ref**} T_i p_1, …, p_m, donde:
  - **ref**, indica que el paso de parámetros se realiza por referencia.
  - T_i, es el tipo (elemental, estructurado o definido por el usuario) de los parámetros formales del *i*-ésimo grupo.
  - p_j ($1 \leq j \leq m$), es el identificador del *j*-ésimo parámetro formal del *i*-ésimo grupo.
- *T* es el tipo de dato que devuelve la función.
- **var**, inicia el grupo de declaraciones de variables locales de la función (si existen).
- **retornar**, es una operación que devuelve el valor de salida resultante de la función.

Ejemplo:

```
{ Calculo del factorial de n:
  n! = n * (n-1) * ... * 2 * 1 }
func factorial(entero n): entero
var
    entero fn, i
inicio
    fn ← 1
    para i ← 2 hasta n en 1 hacer
          fn ← fn * i
    fpara
    retornar(fn)
ffunc
```

5.2. Procedimientos

Los procedimientos son operaciones que pueden recibir datos de entrada, se usan para un fin determinado, y no producen ninguna salida, sin embargo, pueden devolver resultados si entre sus parámetros formales existe alguno declarado por referencia.

La definición de un procedimiento es la siguiente:

```
proc nombre([F1; ...; Fn])
[var]
inicio
    // Instrucciones
fproc
```

Donde:

- *nombre*, es el identificador de la función.
- Cada F_i ($1 \le i \le n$), es un grupo de parámetros formales de la siguiente forma: {**ref**} T_i p_1, …, p_m, donde:
  - **ref**, indica que el paso de parámetros se realiza por referencia.
  - T_i, es el tipo (elemental, estructurado o definido por el usuario) de los parámetros formales del *i*-ésimo grupo.
  - p_j ($1 \le j \le m$), es el identificador del *j*-ésimo parámetro formal del *i*-ésimo grupo.
- **var**, inicia el grupo de declaraciones de variables locales del procedimiento (si existen).

Ejemplo:

```
// Impresión de la constante PI
proc imprimir_pi()
inicio
    escribir("3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190 91")
fproc
```

6. Partes de la Definición de una Acción Nominada

La definición de una acción nominada está formada por dos elementos: La cabecera y el cuerpo.

`func nombre([F1; ...; Fn]): T`	← **Cabecera** →	`proc nombre([F1; ...; Fn])`

`[var]`
`inicio`
`  // Instrucciones`
`  retornar(expr)`
`ffunc`
	← **Cuerpo** →	`[var]`
		`inicio`
		`  // Instrucciones`
		`fproc`

7. Llamada a una Acción Nominada

Una acción nominada previamente definida en un algoritmo puede ser utilizada mediante una llamada. Una llamada esta formada por el nombre de la función o procedimiento con sus parámetros reales.

Ejemplo:

```
resultado ← factorial(5)
```

8. Parámetros

Un parámetro es un tipo especial de variable utilizada por una subrutina para referirse a uno de sus datos de entrada.

Existen dos tipos de parámetros: Formales y reales.

8.1. Parámetros Formales

Los parámetros formales son las variables presentes en la declaración de una acción nominada. Estas variables tomarán los valores de los parámetros reales una vez que sea invocada la acción nominada. Estos parámetros pueden ser de dos tipos:

Parámetros por Valor

Son aquellos parámetros cuyo valor no es modificado por la subrutina. Se les llama también parámetros de entrada. Los parámetros de una función o procedimiento son por defecto parámetros por valor.

Ejemplo: La variable entero es un parámetro formal por valor.

```
func factorial(entero n): entero
```

Parámetros por Referencia

Son aquellos parámetros cuyo valor es modificado por las instrucciones de la subrutina, y son devueltos como resultado. También son llamados parámetros de salida. Para declarar un parámetro por referencia se utiliza la palabra reservada **ref**.

Ejemplo: Este procedimiento devuelve los dos últimos dígitos de un número por referencia.

```
proc ultimos_dos_digitos(entero num; ref entero d1, d2)
```

8.2. Parámetros Reales

Los parámetros reales, también llamados parámetros actuales o argumentos, son aquellos valores que se pasan al realizar una llamada a una subrutina. Estos valores luego son copiados en los parámetros formales de la subrutina una vez que esta se ejecuta.

Ejemplos:

```
resultado ← factorial(5)
ultimos_dos_digitos(1024, a, b)
```

9. Prototipo de una Acción Nominada

El prototipo de una acción nominada es su cabecera o declaración. Dentro de un algoritmo se pueden colocar los prototipos de las acciones nominadas que se van a utilizar antes del algoritmo principal que comienza con la palabra reservada **algoritmo**, para luego definirlos después del **fin** del mismo. La definición de prototipos no es obligatoria, pero ayuda a la legibilidad del código.

Ejemplo:

Sin Prototipos

```
algoritmo calculo_de_factoriales

func factorial(entero n): entero
var
    entero fn, i
inicio
    fn ← 1
    para i ← 2 hasta n en 1 hacer
       fn ← fn * i
    fpara
    retornar(fn)
ffunc

inicio
    var
       entero num, fact
       caracter opc
    escribir("Calculo de factoriales")
    hacer
       escribir("Introduzca un número")
       leer(num)
       fact ← factorial(num)
       escribir("El factorial es: ", fact)
       escribir("Calcular otro factorial? (s/n)")
       leer(opc)
    mientras(opc = 's' ∨ opc = 'S')
fin
```

Con Prototipos

```
algoritmo calculo_de_factoriales

func factorial(entero n): entero

inicio
    var
       entero num, fact
       caracter opc
    escribir("Calculo de factoriales")
    hacer
       escribir("Introduzca un número")
       leer(num)
       fact ← factorial(num)
       escribir("El factorial es: ", fact)
       escribir("Calcular otro factorial? (s/n)")
       leer(opc)
    mientras(opc = 's' ∨ opc = 'S')
fin

func factorial(entero n): entero
var
    entero fn, i
inicio
    fn ← 1
    para i ← 2 hasta n en 1 hacer
       fn ← fn * i
    fpara
    retornar(fn)
ffunc
```

10. Referencias

Martínez, A., Rosquete, D. (2009). NASPI: Una Notación Estándar para Programación Imperativa. Télématique.