UNIVERSIDAD DE CARABAOBO
FACYT - DEPARTAMENTO COMPUTACIÓN
ALGORITMOS Y PROGRAMACIÓN I

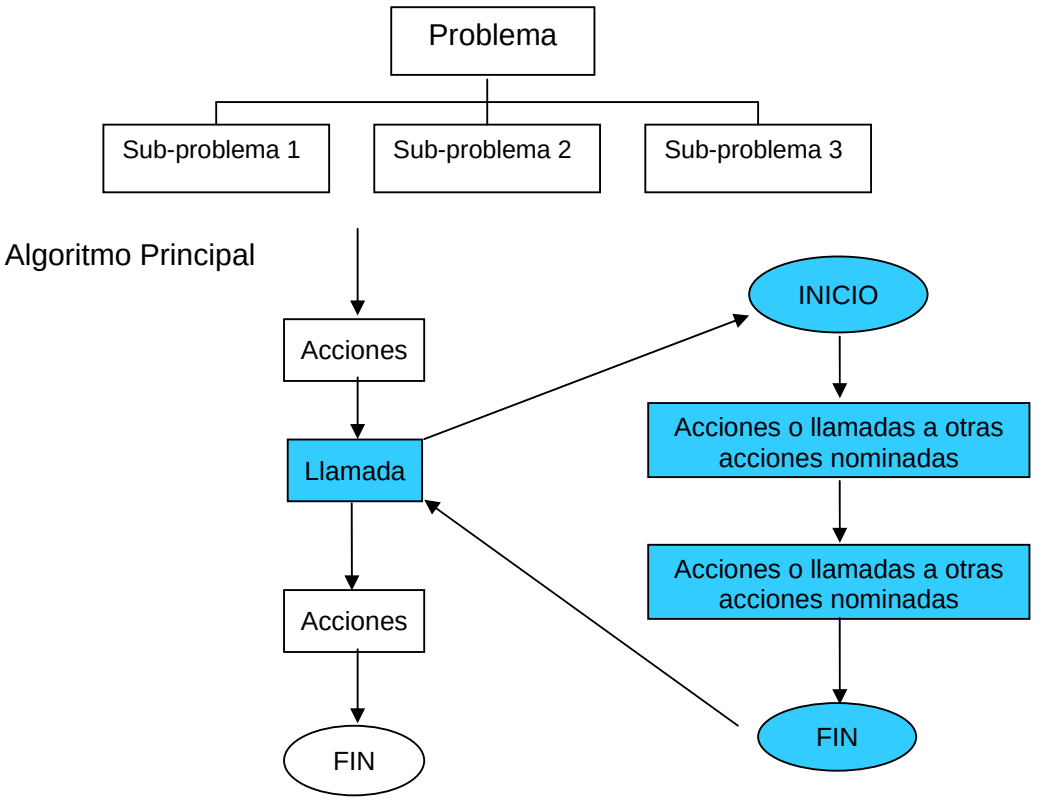
Acciones Nominadas
(Funciones y procedimientos)

Definición:

Son acciones o instrucciones dentro un algoritmo que invocan la ejecución de otro algoritmo (llamado sub-algoritmo) para resolver un sub-problema del problema original que resuelve el primer algoritmo. Una vez definido un sub-algoritmo, éste puede ser invocado o llamado por cualquier otro algoritmo (o sub-algoritmo) tantas veces como se requiera con datos de entrada diferentes, sin tener que repetir el conjunto de acciones del sub-algoritmo. A nivel de programación a las acciones nominadas también se les denomina sub-programas.

Ventajas

- Permite dividir un problema en sub-problemas (problemas más pequeños, más fáciles de resolver), los cuales pueden ser definidos y comprobados individualmente. Esta técnica se conoce como *divide y vencerás*.
- Una vez refinada y comprobada una acción nominada, se puede utilizar varias veces en distintos algoritmos, ahorrando en consecuencia tiempo de programación y evitando la duplicación de código.
- Facilita el desarrollo de un algoritmo o programa entre varias personas, dado que cada sub-algoritmo se puede diseñar independientemente.
- Facilita la creación de algoritmos correctos y la verificación de errores.



Características:

- Los algoritmos se descomponen en un algoritmo principal y cero o más acciones nominadas.
- Toda acción nominada tiene una definición (el algoritmo en sí) y una o más llamadas a ejecución desde el algoritmo principal o cualquier acción nominada. La definición debe aparecer antes de cualquiera de las llamadas.
- Cada acción nominada utiliza sus propias variables, llamadas variables locales. También puede definir constantes y tipos locales.
- El algoritmo que invoca la acción nominada le comunica los datos de entrada asignando valores a un conjunto especial de variables locales de la acción nominada llamados parámetros. Los resultados son entregados al algoritmo invocador a través de parte de los parámetros y/o como el resultado de la evaluación de la acción nominada, al estilo de las funciones matemáticas.

Ámbito de variables: variables locales y globales

El ámbito de las variables se refiere a la duración de su existencia, es decir, en qué puntos de un algoritmo éstas pueden ser accedidas (consultadas o modificadas). Existen dos tipos de variables según su ámbito:

Variables locales: son aquellas que están declaradas dentro de una acción nominada y sólo son accesibles dentro de la misma. Estas variables son distintas de las variables que puedan existir con el mismo nombre declaradas en el algoritmo principal u otra acción nominada.

Variables globales: son aquellas que están declaradas en el algoritmo principal. Éstas pueden ser accedidas en cualquier parte del algoritmo principal y dentro de todas las acciones nominadas. En caso de que una acción nominada tenga una variable local con el mismo nombre de una variable global, sólo la local es accesible dentro de la acción nominada.

Parámetros y argumentos

Para que un algoritmo invoque o llame a una acción nominada debe indicar el nombre de la acción y los valores de entrada para la misma. Opcionalmente puede indicarle variables en las que desea que se guarden algunos o todos los resultados de la acción nominada. Esto se realiza a través de los argumentos y los parámetros:

Argumentos: También llamados parámetros reales, son los valores o variables que un algoritmo especifica como parte de la llamada a una acción nominada, bien sea para que sirvan como datos de entrada, para guardar datos de salida de la acción, o ambos.

Parámetros: También llamados parámetros formales, son variables locales de la acción nominada que obtienen automáticamente los valores de los argumentos indicados por el algoritmo invocador.

Paso de parámetros

La manera como se relaciona cada argumento con su correspondiente parámetro viene dada por el método de *paso de parámetros*. Existen dos métodos:

Por valor (entrada): En este caso el argumento sólo sirve como dato de entrada y puede ser una constante, una variable o, en general, una expresión. El parámetro recibe automáticamente una copia del argumento (como si se realizara una asignación). Esto implica que en caso de que el argumento sea una variable, las modificaciones que se hagan al parámetro dentro de la acción nominada no afectan a la variable del argumento.

Por referencia (entrada/salida): En este caso el argumento sirve al mismo tiempo como dato de entrada y como dato de salida, de modo que sólo puede ser una variable. Dentro de la acción nominada el parámetro *hace referencia* a la misma posición de memoria que el argumento. Esto implica que cualquier cambio hecho al parámetro afecta de igual forma al argumento, y viceversa en caso de que la variable del argumento sea visible en la acción nominada. El algoritmo que invoca la llamada puede entonces utilizar cualquier resultado almacenado en la variable del argumento después de la ejecución de la acción nominada.

Clasificación de las acciones nominadas

Existen dos tipos de acciones nominadas, según la manera como se manejan sus llamadas: las funciones y los procedimientos:

- **Funciones:** Las llamadas a funciones se consideran expresiones, y por lo tanto deben dar un valor como resultado de evaluación. Pueden aparecer en cualquier parte donde pueda aparecer una expresión: al lado derecho de una asignación, dentro de otra expresión, como argumento en la llamada de otra acción nominada, etc. En el algoritmo de la función se debe especificar el valor que da como resultado la función (valor de retorno), lo cual se hace a través de la instrucción *retornar*, la cual va seguida de una expresión que indica el valor a ser retornado. Al ejecutarse una instrucción *retornar* se termina la ejecución de la función, retornando al algoritmo que la invocó.

  Sintaxis de declaración:

  ```
  func <nomFunc>(<param1>: <Tipo1>, ... , <paramn>: <Tipon>): <TipoRet>
  <declaraciones de contantes, tipos y variables locales>
  inicio
    <cuerpo de la función (algoritmo)>
    retornar <expresión de retorno>
  fin
  ```

  Donde:
  - <nomFunc> es un identificador que identifica la función.
  - <param$_i$> es el i-ésimo parámetro de la función. De forma predeterminada el paso de parámetros es por valor. Para especificar que un parámetro es por referencia se antepone `var` o `ref` antes del nombre del parámetro.
  - <Tipo$_i$> es el tipo del i-ésimo parámetro.
  - <TipoRet> es el tipo del valor de retorno. Éste sólo puede ser un tipo primitivo.

Ejemplo de declaración:

```
func suma(x: Entero, y: Entero): Entero
inicio
  retornar x + y
fin
```

Sintaxis de llamada:

```
<nomFunc>(<arg1>, <arg2>, ... , <argn>)
```

Donde:
- <nomFunc> es el identificador de la función.
- <arg_i> es el i-ésimo argumento de la llamada a la función.

Ejemplos de llamada:

```
res ← suma(8, 10)  // Forma correcta, argumentos constantes
res ← suma(2*r, 10 div m) // Forma correcta, si r y m son enteros
escribir("a + b = ", suma(a, b)) // Forma correcta,  si a y b son enteros
res ← 4 * suma(3, 7) // Forma correcta
res ← suma (10.6 , 6) // Forma incorrecta (10.6 no es Entero)
suma(x , 10) // Forma incorrecta (se está usando como instrucción)
```

- **Procedimientos:** Las llamadas a procedimientos se consideran instrucciones, y por lo tanto no retornan ningún resultado. Pueden aparecer en cualquier parte donde pueda aparecer una instrucción.

Sintaxis de declaración:

```
proc <nomProc>(<param1>: <Tipo1>, ... , <paramn>: <Tipon>)
<declaraciones de contantes, tipos y variables locales>
inicio
  <cuerpo del procedimiento (algoritmo)>
fin
```

Donde:
- <nomProc> es un identificador que identifica el procedimiento.
- <$param_i$> es el i-ésimo parámetro del procedimiento. De forma predeterminada el paso de parámetros es por valor. Para especificar que un parámetro es por referencia se antepone `var` o `ref` antes del nombre del parámetro.
- <$Tipo_i$> es el tipo del i-ésimo parámetro.

Ejemplo de declaración:

```
proc división(x: Entero, y: Entero, var coci: Entero, var resto: Entero)
inicio
  coci ← x div y
  resto ← x mod y
fin
```

Sintaxis de llamada:

```
<nomProc>(<arg1>, <arg2>, ... , <argn>)
```

Donde:

- <nomProc> es el identificador del procedimiento.
- $<arg_i>$ es el i-ésimo argumento de la llamada al procedimiento.

Ejemplo de llamada:

```
división(10, 4, a, b)
escribir("Cociente: ", a, "; Resto: ", b)
```

Ejemplo completo

Algoritmo que calcula una aproximación de la función seno a partir de la serie:

$$sen(x) = \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

para algún *n* dado, asumiendo que no se tiene un operador de potenciación disponible:

```
func factorial(n: Entero): Entero
var
  r, i: Entero
inicio
  r ← 1
  para i ← 1 hasta n hacer
    r ← r * i
  fpara
  retornar r
fin
```

```
func potencia(x: Real, y: Entero): Real
var
  i: Entero
  r: Real
inicio
  r ← 1.0
  para i ← 1 hasta y hacer
    r ← r * x
  fpara
  retornar r
fin

func seno(x: Real, n: Entero): Real
var
  i: Entero
  r: Real
inicio
  r ← 0.0
  para i ← 0 hasta n hacer
    r ← r + (potencia(-1.0, i) / factorial(2*i+1)) * potencia(x, 2*i+1)
  fpara
  retornar r
fin

Algoritmo cálculoSeno
var
  x: Real
inicio
  escribir(“Indique un valor para x: “)
  leer(x)
  escribir(“sen(x) = “, seno(x, 5000))
fin
```