

TEMA 4

Diseño descendente

1. Introducción.

La resolución de problemas complejos se facilita si se dividen en problemas más pequeños. La solución de estos subproblemas se realiza mediante subalgoritmos: acciones o funciones. El uso de subalgoritmos permite desarrollar algoritmos de problemas complejos utilizando la metodología de diseño descendente. Los subalgoritmos son módulos que están diseñados para ejecutar alguna tarea específica. Se escriben solamente una vez, pero pueden ser utilizados en diferentes puntos del algoritmo, referenciados por otros subalgoritmos, e incluso por ellos mismos (recursividad).

La modularización proporciona dos importantes ventajas:

- **Evitar la escritura repetida de instrucciones**, ya que una vez definido un módulo de un algoritmo, éste puede ser utilizado desde otros puntos del algoritmo. Cada vez que el módulo es invocado, puede procesar un conjunto distinto de datos.
- **Más claridad**. Cada módulo representa una parte bien definida del problema total. Al modular el algoritmo, éste se estructura en tareas más simples.

Vamos a distinguir dos tipos de módulos:

- Acciones.
- Funciones.

2. Acciones.

Se presenta el concepto de acción como un mecanismo de abstracción que permite enriquecer el lenguaje algorítmico con la creación de instrucciones "virtuales" más potentes que las primitivas.

Una acción es un subalgoritmo que ejecuta un proceso específico. La declaración de una acción sirve para definir un conjunto de instrucciones y asociarlas con un identificador. El identificador o nombre de la acción servirá para hacer referencia a esta nueva "instrucción virtual".

2.1. Definición de acciones.

En la definición de una acción vamos a seguir el siguiente formato:

- 1.- Cabecera
- 2.- Especificación
- 3.- Cuerpo

La **cabecera** está formada por la palabra 'acción' seguida del identificador o nombre de la acción y a continuación la lista de parámetros.

```
acción NombreAcción ( declaracion de parámetros )
```

Una acción tiene una lista de parámetros en su definición, que son los denominados **parámetros formales**, que tienen un carácter local (solo están definidos y por tanto pueden utilizarse dentro de la acción). (En algunos casos, esta lista puede ser vacía). Los parámetros se utilizan como herramienta para comunicar datos y resultados entre quien invoca la acción y ésta.

En esta lista se indica para cada parámetro, su **nombre**, **tipo** y **clase** de la siguiente forma:

- **Tipo:** se refiere al tipo de dato que representa el parámetro. Se especifican de la misma forma que se declaran las variables:

tipoDeDato nombreParámetro

- **Clase:** se especifica la clase de parámetro según el siguiente convenio:

- Los parámetros de entrada van precedidos por la palabra clave **E/**.
Se usan para introducir datos en la acción (entrada de datos)
- Los de salida, por la palabra clave **S/**.
Se usan para devolver resultados de la acción (salida de rdos)
- Los de entrada/salida, por la palabra clave **E/S**.

Funcionan como parámetros de entrada cuando se llama (se invoca, se activa) la acción, y como salida cuando la acción acaba su ejecución.

(Más adelante explicamos algunos detalles más sobre los parámetros)

El **cuerpo** comprende la parte de declaraciones e instrucciones que constituyen la acción.

Si los datos de una llamada a una acción cumplen lo exigido en la especificación de la acción, entonces tras la llamada, se cumplirá lo previsto para el efecto producido indicado en la especificación.

Al efectuarse la llamada cobran sentido las variables declaradas dentro de la acción. Estas se denominan **variables locales**: su ámbito de existencia queda restringido a la propia acción y solamente durante la ejecución de ésta. Aunque en otro punto del algoritmo haya variables con el mismo nombre, éstas y aquéllas deben considerarse objetos distintos.

Ejemplo:

Especifica y diseña un algoritmo que dibuje un rectángulo de una determinada base y altura mediante los símbolos '+', '-' y '|', por ejemplo, para una base = 7 y una altura = 4 sería:

```

+-----+
|       |
|       |
+-----+

```

Especificación:

Interfaz:

 entrada: entero base, entero altura

 salida:

Efecto:

 Condiciones de entrada:

 base, altura >=2

 Efecto producido:

 dibuja un rectángulo...

algoritmo dibujaRectangulo

variables

 entero base, altura, i, j

principio

 leer(base)

 leer(altura)

 //dibujar línea superior

 escribir('+')

para i=1 **hasta** (base-2) **hacer**

 escribir('-')

fpara

 escribir('+')

 escribirln

 //Dibujar líneas intermedias

para j=1 **hasta** (altura-2) **hacer**

 escribir('|')

para i=1 **hasta** (base-2) **hacer**

 escribir(' ')

fpara

 escribirln('|')

fpara

 //Dibujar línea inferior

 escribir('+')

para i=1 **hasta** (base-2) **hacer**

```

        escribir('-')
    fpara
        escribirln('+')
    fin

```

El algoritmo anterior sería más simple si dispusiéramos de acciones (instrucciones) más potentes, en concreto si tuviéramos una acción capaz de dibujar una línea de la forma ++++++.

Esa acción no es una primitiva de nuestro lenguaje, por tanto tendremos que construirla. Es la siguiente:

acción dibujaLinea(E/ entero ancho)
--

//PRE: ancho >=2

//POS: dibuja una línea de la forma +-----+

variables entero i principio escribir('+') i=0 mientras que (i< ancho-2) hacer escribir('-') i=i+1 fmq escribirln('+') fin
--

Se han recuadrado cada una de las partes de una acción, recuerda:

- 1.- Cabecera
- 2.- Especificación
- 4.- Cuerpo

Observa que la lista de parámetros contiene un parámetro de tipo entero llamado ancho. Permite comunicar a la acción la longitud de la línea a pintar.

Una vez declarada la acción, se dispone de una nueva instrucción, una instrucción virtual, que puede utilizarse tantas veces como sea necesaria (como si

fuese una instrucción primitiva más del lenguaje de programación). La ejecución de esta “instrucción” se realizará mediante una **llamada** (invocación) a la acción.

Así, nuestro algoritmo `dibujaRectangulo` podría escribirse de la siguiente manera, utilizando la acción `dibujaLinea` que acabamos de crear:

Especificación:

Interfaz:

 entrada: entero base, entero altura

 salida:

Efecto:

 Condiciones de entrada:

 base, altura ≥ 2

 Efecto producido:

 dibuja un rectángulo...

No cambia. ¡Es el mismo programa!

algoritmo `dibujaRectangulo`

variables

 entero base, altura, i, j

principio

 leer(base)

 leer(altura)

 //dibujar línea superior

`dibujaLinea(base)`

 //Dibujar líneas intermedias

para j=1 **hasta** (altura-2) **hacer**

 escribir('|')

para i=1 **hasta** (base-2) **hacer**

 escribir(' ')

fpara

escribirln('|')

fpara

 //Dibujar línea inferior

`dibujaLinea(base)`

fin

Sustituye a
escribir('+')
para i=1 **hasta** (base-2)
hacer
 escribir('-')
fpara;
escribirln('+')

Ventaja: algoritmo más simple, gracias al uso de “instrucciones” más potentes (`dibujaLinea`)

2.2 Parámetros.

Parámetros formales

En ocasiones (la mayoría de las veces) interesa que una acción haga un determinado proceso a partir de ciertos datos de entrada, obteniendo unos resultados. Para ello se utilizan los **parámetros**. Es decir, los parámetros se usan para intercambiar información (datos de entrada y resultados) entre el algoritmo que llama a la acción y éste.

Una acción tiene una lista de parámetros en su cabecera denominados **parámetros formales**. Un parámetro formal es una variable que sólo se utiliza dentro de la acción. Tiene por lo tanto un carácter local a ésta. (En el ejemplo que se muestra a continuación son parámetros formales “ext”, “cent” y “longitud”).

Ejemplo:

Supongamos que en el ejemplo anterior, necesitamos una acción para dibujar una línea, pero esta vez de cualquier longitud y con cualquier pareja de caracteres. En este caso la acción se escribe así:

```
acción dibujaLinea(E/carácter ext ;E/carácter cent; E/entero
longitud)
//PRE: longitud>0
//POS: dibuja una línea de tamaño longitud con el carácter
//cent de relleno y el carácter ext en los extremos
Variables
    entero i
principio
    escribir (ext)
    i=0
    mientras que i < longitud-2 hacer
        escribir(cent)
        i=i+1
    fmq
    escribirln (ext)
fin
```

Tanto **ext** y **cent** como **longitud** están declarados de entrada, eso significa que la acción `dibujaLinea` los utiliza para recibir datos (desde su punto de vista, esos datos **entran** en su ámbito). Cuando “alguien” invoque la acción deberá suministrarle valores para los tres (más adelante explicamos cómo), y en función de esos valores dibujará una línea con una determinada longitud y utilizando unos determinados caracteres.

2.1.1 Clases de parámetros formales.

El intercambio de información entre la acción y el algoritmo que la llama se realiza por medio de los parámetros. Distinguimos varios tipos de parámetros formales dependiendo del sentido del flujo de la información:

- **Parámetros de entrada:**

El sentido de flujo de la información es solo de entrada, desde el punto de vista de la acción. Cuando se produce la llamada a la acción, el valor de cada parámetro real se asigna a su parámetro formal correspondiente.

Tras la ejecución de la acción puede que cambie el valor del parámetro o de los parámetros formales pero esto ya no le afectará a los parámetros reales correspondientes.

- **Parámetros de salida:**

Son aquellos parámetros cuyo valor a la hora de comenzar la ejecución de la acción es irrelevante y que una vez ejecutado éste, asignan el valor resultante de la ejecución de la acción a su correspondiente parámetro real.

- **Parámetros de entrada/salida:**

Son aquéllos cuyo valor inicial es un dato necesario para la ejecución de la acción, por tanto, cuando se produce la llamada se asigna el contenido de cada parámetro real a su correspondiente parámetro formal. Además pueden modificar su valor dentro de la acción y este valor es el que se devuelve al algoritmo desde el que se ha llamado a la acción (asignándolo a su correspondiente parámetro real que, como se explica a continuación, aparecerá en la llamada).

2.3. Llamada a una acción.

Cuando se quiere que las instrucciones descritas dentro de una acción sean ejecutadas, hay que **llamar** (invocar) a esa acción. Esto se hace escribiendo el nombre de ésta y una lista de parámetros, que son los **parámetros reales**. Cada **parámetro real** se empareja con el **parámetro formal** que ocupa su misma posición en la declaración de la acción. Ambas listas de parámetros deben coincidir en:

- número de parámetros
- tipo de los parámetros

Sintaxis de una instrucción de **llamada** a una acción:

`nombreAcción (lista de parámetros reales)`

Los parámetros reales, cuando se asocian con parámetros formales **de entrada**, pueden ser no sólo variables, sino **expresiones** del tipo especificado en la declaración (ya que no se va a producir una asignación del parámetro formal al real a la finalización de la acción). En todos los demás casos han de ser **variables**.

Ejemplo:

Supongamos que en un punto de un programa se necesita dibujar una línea de longitud 5 como las bases del rectángulo. La instrucción de llamada a la acción sería:

```
dibujaLinea('+','-', 5);
```

Al ejecutarse la llamada sucede lo siguiente:

- 1) Se evalúan los parámetros reales (expresion1, ..., expresionN)
- 2) Se asigna el valor de <expresion_i> al parámetro formal i-ésimo de la acción *nombreAcción*. i=1, ..., N.
- 3) Se ejecuta la acción *nombreAcción*.

Al producirse la llamada a una acción, las variables declaradas dentro de ésta cobran sentido. Estas se denominan **variables locales**: su ámbito queda restringido a la propia acción y solamente durante la ejecución de ésta. Por tanto, es posible que en otro punto del algoritmo haya variables con el mismo identificador, ya que éstas se consideran objetos diferentes.

Ejemplo:**Especificación:**

Interfaz:

entrada: entero base, entero altura

salida:

Efecto:

Condiciones de entrada:

 base, altura ≥ 2

Efecto producido:

dibuja un rectángulo...

algoritmo dibujaRectangulo**variables**

entero base, altura, i, j

principio

leer(base)

leer(altura)

//dibujar línea superior

dibujaLinea('+', '-', base)

//Dibujar líneas intermedias

para j=1 **hasta** (altura-2) **hacer**

dibujaLinea('|', ' ', base)

fpara

//Dibujar línea inferior

dibujaLinea('+', '-', base)

fin**acción** dibujaLinea(E/carácter ext ;E/carácter cent; E/entero longitud)

//Especificación anterior.

Variables

entero i

principio

escribir (ext)

i=0

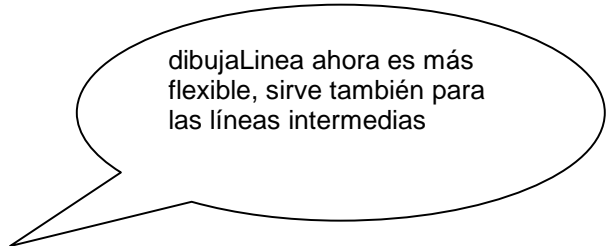
mientras que i < longitud-2 **hacer**

escribir(cent)

i=i+1

fmq

escribirln (ext)

fin


dibujaLinea ahora es más flexible, sirve también para las líneas intermedias

Ejemplo: Algoritmo que toma dos enteros y los ordena de forma creciente.

Especificación:

Interfaz:

 entrada: entero a,b

 salida:

Efecto:

 Condiciones de entrada:

 Efecto producido:

$a \geq b$

algoritmo orden

variables

 entero a,b

principio

 leer(a, b)

 ordenar(a,b)

 escribir(a,b)

fin

acción ordenar(E/S entero x,y)

variables

 entero aux

principio

si $x > y$ **entonces**

 aux = x

 x = y

 y = aux

fsi

fin

En este caso, x e y son parámetros de entrada/salida, significa que cuando se produce la llamada actúan como entrada, reciben datos, y cuando finaliza la ejecución de la acción funcionan como salida, devuelven resultados

3. Funciones.

Una función es otro tipo de subprograma o módulo. Constituye un mecanismo de abstracción que permiten definir cálculos virtuales parametrizados.

La diferencia fundamental con las acciones, además de su diferente declaración y su diferente forma de ser invocada como se verá a continuación, estriba en que **una función devuelve siempre un único valor** mientras que una acción puede devolver ninguno, uno o varios valores como ya se ha visto.

Toda función que se utilice en un algoritmo tiene que haberse definido previamente. Una función puede estar predefinida de forma estándar (sin, cos, sqr, abs,...) o encontrarse en una biblioteca de funciones o bien ser creada por el programador.

3.1. Llamada a una función.

Ejemplos de utilización de funciones en un algoritmo son:

- 1) **si** exp(x) > 1 **entonces**

 si no

 fsi
- 2) d=sucesor(d)
- 3) **para** i=1 **hasta** sqrt(j)+1 **hacer**

 fpara
- 4) escribir(sin(x)+cos(x)/2)

En estos ejemplos aparecen funciones que hemos utilizado ya a lo largo del curso. Estas están predefinidas (funciones estándar) pero la manera de utilizarlas (llamarlas) es idéntica que para las funciones definidas por el usuario.

Una función se llama por tanto escribiendo su identificador y la lista de parámetros reales. Una llamada a una función aparece como parte de una expresión.

Una vez llamada la función se ejecutan las instrucciones asociadas en su definición. Al terminar la ejecución de las instrucciones de la función se continúa en el punto en que está la llamada, sustituyéndose esta por el valor generado por la

función. Es decir, una llamada a una función equivale, a todos los efectos, al valor devuelto por dicha función.

3.2. Declaración de funciones.

Una función es un subalgoritmo, eventualmente parametrizado, que **devuelve un único valor**.

Una función se declara describiendo el algoritmo de cálculo correspondiente, incluyendo al menos una acción que indique el valor a devolver por la función.

Sintaxis:

Como se ha indicado, una función no es más que un algoritmo, por tanto, su estructura será la ya conocida, a excepción de:

- su **cabecera**, que se escribe así:

```
función nomFuncion(parám formales) devuelve tipoDeDato
```

donde la lista de parámetros formales incluye todos los parámetros de entrada sin indicar la clase (no es necesario ya que son todos de entrada), y en tipoDeDato se especifica el tipo de dato del único valor que devuelve la función.

- su **cuerpo**, en el que para indicar el valor que devuelve la función se utilizará la palabra “devuelve” de la forma:

```
devuelve( expresión )
```

Luego, la estructura de una función queda:

```
función nomFuncion (parám formales ) devuelve tipoDeDato
//PRE: Precondición
//POS: Poscondición
declaraciones (variables, tipos, ...)
principio
    instrucciones
    devuelve( expresión )
fin
```

Notas:

- 1) Una función devuelve siempre un único valor en el propio nombre de función, por tanto, los parámetros formales que utiliza son de entrada (salvo excepciones impuestas en algún lenguaje de programación).
- 2) Dentro del cuerpo de la función deberá ejecutarse una y solo una instrucción “devuelve”. (Eventualmente podrán aparecer varias instrucciones “devuelve” pero se deberá tener la certeza de que en cada llamada a la función solo se ejecutará una).

Ejemplos:

- 1.- Función que calcula el valor de la tangente de un ángulo.

```

función tg(real x) devuelve real
//PRE:  $x \neq \pi/2 + k\pi$ 
//POS: devuelve  $\text{sen}(x)/\text{cos}(x)$ 
principio
    devuelve( $\text{sen}(x)/\text{cos}(x)$ )
fin

```

- 2.- Función esPar.

```

función esPar(entero n) devuelve booleano
//PRE: n valor cualquiera
//POS: Devuelve el valor verdad si n es par o
        devuelve el valor falso si n es impar
principio
    si  $n \bmod 2 == 0$  entonces
        devuelve( verdad )
    si no
        devuelve( falso )
    fsi
fin

```

- 3.- Diseñar una función que devuelva la potencia de un valor real positivo elevado a un exponente entero positivo.

```
Función potencia(real base; entero exponente) devuelve real
//PRE: base>=0
//POS: Devuelve base^exponente
variables
    entero i
    real resultado
principio
    resultado=1
    i = 1
    mientras que i<=exponente hacer
        resultado = resultado * base
        i = i + 1
    fmq
    devuelve(resultado)
fin
```


4- Algoritmo que calcule un número combinatorio utilizando una función para el cálculo del factorial.

Especificación:

Interfaz:

 entrada: entero m, entero n

 salida: entero comb

Efecto:

 Condiciones de entrada:

$m > 0$, $n > 0$, $m > n$

 Efecto producido:

 combinatorio = $m! / (n! * (m-n)!)$

algoritmo combinatorio

variables

 entero m, n, comb

principio

 leer(m,n)

 comb= factorial(m) DIV (factorial(n) * factorial(m-n))

 escribir(comb)

fin

funcion factorial(entero x) **devuelve** entero

//PRE: $x > 0$

//POS: devuelve $x!$

variables

 entero f, i

principio

 f=1

para i=1 **hasta** x **hacer**

 f= f * i

fpara

 devuelve(f)

fin

Ejercicios propuestos

1. Especificar y diseñar un algoritmo que genere las tablas de multiplicar de los enteros n al m (n y m se leerán por el teclado). Dicho algoritmo utilizará una acción *tabla* (diseñala también) que a partir de un número n entero positivo escribe su tabla de multiplicar.
2. Construir una función que calcule el máximo común divisor de dos números enteros positivos. Recuerda: $\text{mcd}(a, 0) = a$; $\text{mcd}(a, b) = \text{mcd}(b, \text{resto}(a/b))$
3. Reducir una fracción dada mediante dos variables, para el numerador y el denominador, utilizando la función anterior. Debe devolver otras dos variables con los valores del numerador y denominador de la fracción reducida.
4. Acción que Intercambie los valores de dos variables de tipo real.
5. Algoritmo que ordene los valores de tres variables reales en orden creciente utilizando la acción anterior.
6. Acción que devuelva el área y el perímetro de un círculo dado el radio.
7. Función que convierta grados Fahrenheit en Celsius, $\text{Cel} = (5/9) (\text{Fahr} - 32)$.
8. Dado un número entero construir una función que nos devuelva el número de cifras que tiene.
9. Algoritmo que calcule el área de una corona circular. Utilizar una función que devuelva el área de un círculo de radio dado (escribe también esta función).
10. Acción que calcule el cociente y el resto de la división entera (sin utilizar DIV ni MOD).
11. Función que determine si un número entero es o no un número primo.
12. Acción que realice la conversión de coordenadas polares (r, \varnothing) a coordenadas cartesianas (x, y) .

Nota: $x = r \cos(\varnothing)$, $y = r \sin(\varnothing)$.

13. Completa el siguiente algoritmo.

Especificación:

Interfaz:

Entrada: real x, y entero n

Salida: real rdo

Efecto:

Condiciones de entrada: x, y cualesquiera, $n > 0$ }

Algoritmo newton

```

.....
principio
  leer(x, y)
  repetir
    leer(n)
  mientras que .....
    i=0
    rdo=.....
  mientras que ..... hacer

    rdo= ....combinatorio.....potencia.....potencia.....

    i= .....
  fmq
    escribir(rdo)
fin

..... potencia .....
//PRE: base cualquiera,exponente >= 0
//POS: devuelve baseexponente
variables

.....

principio
  pot = .....
  para i=1 hasta exponente hacer
    pot= pot * base
  fpara

  .....
fin

función combinatorio(entero a, b) devuelve real
//PRE: a, b >= 0, a >= b
//POS: devuelve el combinatorio a sobre b
.....
fin

```