Algoritmos: componentes

Todo algoritmo se puede construir con sólo 3 componentes estructurales:

- *Secuencia* : grupo de acciones que se ejecutan una tras otra.
- *Selección* : selecciona un camino entre varios según el valor de condición. Si toma el valor *cierto* se realiza una acción (o grupo de acciones) pero si toma el valor *falso* se realiza otra o no se hace nada.
- *Repetición* : repite un conjunto de acciones cierto número de veces.

En un algoritmo concreto puede haber uno, dos o los tres componentes.

SECUENCIA DE *ACCIONES*

```
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café
agregar el café molido al contenedor
enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
...
```

Ejemplos de selección

Se podría considerar la situación en que no hay café molido antes de poner el café.

¿Qué hacer en este caso?

```
si ( está vacío el bote de café molido )
    moler café
fin si
```

Esto responde al esquema

```
si ( CONDICIÓN )
    acción a realizar si CONDICIÓN es cierta
fin si
```

Esto se agrega al algoritmo antes de la línea 'agregar el café molido...' y quedaría así

```
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café
 si ( está vacío el bote de café molido )
   moler café
fin si
agregar el café molido al contenedor
enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
```

Qué es una CONDICIÓN? La podemos ver como una pregunta a la que hay que responder en el momento exacto en que se llega a ella mientras se ejecuta el algoritmo y cuyas respuestas pueden ser (SI, VERDADERO...) o (NO, FALSO etc).

El álgebra de Bool se construye sobre un conjunto que sólo tiene dos valores que se pueden representar así

El valor falso: 0, FALSE, FALSO, NO ...

El valor cierto: 1, TRUE, CIERTO, SI ...

CONDICIÓN: expresión que al evaluarla produce VERDADERO o FALSO.

Otra situación: considerar la cantidad de café según el gusto.

```
si ( quiero el café cargado )
    agregar 4 cucharaditas de café
si no
    agregar 2 cucharaditas
fin si
```

Esto responde al esquema

```
si ( CONDICIÓN )
    acción a realizar si CONDICIÓN es cierta
si no
    acción a realizar si CONDICIÓN es falsa
fin si
```

Esto se agrega EN LUGAR DE LA LÍNEA *agregar café molido...* y queda así

```
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café

si ( está vacío el bote de café molido )
    moler café
fin si

{    línea anulada  agregar el café molido al contenedor}
si ( quiero el café cargado )
    agregar 4 cucharaditas de café
si no
    agregar 2 cucharaditas
fin si

enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
```

Ejercicios

1. Escribir un ejemplo del primer tipo SI .. FIN SI
2. Escribir un ejemplo del segundo tipo SI .. SI NO .. FIN SI

Contar

En muchas ocasiones necesitamos contar. ¿Cómo contamos?
Las personas podemos usar métodos sonoros, gráficos u otros para contar:
contar de viva voz diciendo uno, dos, tres... por cada cosa que se cuenta
podemos hacer una raya con un lápiz por cada cosa
Una forma un poco rara de contar (pero válida) sería
usar una calculadora

poner un cero inicial

por cada cosa a contar sumar 1.

En este ejemplo usamos la memoria que almacena lo que hay en la pantalla para contar las cosas que nos interesan.

Recordemos que los ordenadores son muy buenos manejando números y haciendo operaciones aritméticas. Para ellos es más fácil este último método que cualquiera de los dos primeros: el que utiliza sonidos o el que utiliza gráficos.

La forma de representar esa manera de contar sería esta

Para poner el *valor inicial* (0) en la calculadora

```
calculadora <-- 0
```

Para sumar 1 a lo que hay en pantalla

```
calculadora <-- calculadora + 1
```

Para *contar hasta cinco* se podría hacer así

```
calculadora <-- 0
calculadora <-- calculadora + 1
calculadora <-- calculadora + 1
calculadora <-- calculadora + 1
calculadora <-- calculadora + 1
calculadora <-- calculadora + 1
```

o también así

```
calculadora <-- 1
calculadora <-- calculadora + 1
calculadora <-- calculadora + 1
calculadora <-- calculadora + 1
calculadora <-- calculadora + 1
```

Observe: Para contar 5 cosas podemos hacer tres cuentas equivalentes

Contar desde el 1 hasta el 5

Contar desde el 5 hasta el 1

Contar desde el entero X hasta el entero X + 5

Ejercicios

Si una instrucción se repite n veces póngala la primera vez, la última y entre medio puntos suspensivos así

```
calculadora <-- calculadora + 1  {vez 1}
...                              {veces 2 a 99}
calculadora <-- calculadora + 1  {vez 100}
```

Escribir el seudocódigo que cuenta desde 1 hasta 100 de 2 en 2

Escribir el seudocódigo que cuenta desde 100 hasta cero de 1 en 1

¿Como podemos contar desde 40 hasta 60 de 1 en 1?

En vez de usar los puntos suspensivos para indicar las instrucciones que se repiten se puede expresar así para contar desde 1 hasta 100 de 2 en 2.

```
calculadora <-- 1
mientras (calculadora < 100)
    calculadora <-- calculadora + 2
```

```
fin mientras
```

Así se podría contar desde 100 hasta 1 de uno en uno.

```
calculadora <-- 100
mientras (calculadora > 0)
    calculadora <-- calculadora - 1
fin mientras
```

Así se podría contar desde 40 hasta 60 de uno en uno.

```
calculadora <-- 40
mientras (calculadora < 60)
    calculadora <-- calculadora + 1
fin mientras
```

Ejemplos de repetición

Volviendo al algoritmo del café ¿Cómo describiríamos la operación de agregar cuatro cucharadas de café para hacerlo cargado?

```
ctaCucharadas <-- 0
mientras (ctaCucharadas < 4)
    poner una cucharada de café
    ctaCucharadas <-- ctaCucharadas + 1
fin mientras
```

o también así

```
ctaCucharadas <-- 1
mientras (ctaCucharadas <= 4)
    poner una cucharada de café
    ctaCucharadas <-- ctaCucharadas + 1
fin mientras
```

En ctaCucharadas se anotan las cucharadas PUESTAS. Por eso, por cada cucharada que se pone se incrementa en uno el valor anotado.

También se podría hacer así

```
ctaCucharadas <-- 4
mientras (ctaCucharadas > 0)
    poner una cucharada de café
    ctaCucharadas <-- ctaCucharadas - 1
fin mientras
```

Ahora ctaCucharadas anota las cucharadas pendientes de poner, por eso, por cada una que se agrega se resta 1 a las que faltan. Por eso hubiera sido más claro así

```
faltan Cucharadas <-- 4
mientras (faltanCucharadas > 0)
    poner una cucharada de café
    faltanCucharadas <-- faltanCucharadas - 1
fin mientras
```

El esquema que siguen los tres ejemplos es

```
poner el valor inicial
mientras (CONDICIÓN)
    acción a realizar si CONDICIÓN ES CIERTA
    contar uno más (sumando o restando según proceda)
fin mientras
```

Al final, el algoritmo podría quedar así

```
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café

si (está vacío el bote del café molido)
   Moler café
fin si

si (quiero el café cargado)
    ctaCucharadas <-- 0
    mientras ctaCucharadas < 4
        poner una cucharada
        ctaCucharadas <-- ctaCucharadas + 1
    fin mientras
si no
    ctaCucharadas <-- 0
    mientras ctaCucharadas < 2
        poner una cucharada
        ctaCucharadas <-- ctaCucharadas + 1
    fin mientras
fin si

enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
```

Recapitulando

¿Qué hemos aprendido?

Todo algoritmo se representa mediante frases o sentencias

Cada frase o sentencia representa una acción que tendrá que realizar el procesador del algoritmo.

Las frases o sentencias sólo se pueden agrupar de tres formas: secuencia, selección e iteración

- La secuencia sólo tiene un formato: sentencia, sentencia, sentencia ...
- La selección puede ser SI .. FIN SI, o bien SI .. SI NO .. FIN SI
- La iteración tiene (por ahora) un formato: MIENTRAS .. FIN MIENTRAS

¿Nos ha mostrado algo más el ejemplo?

¡¡¡SI!!!

Para comprenderlo veamos cómo lo hemos resuelto.

Hemos empezado por aquí

```
{1 etapa/refinamiento}
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café
agregar el café molido al contenedor
enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
...
```

Después hemos considerado que si no hay café molido no se puede agregar el café y hemos llegado a esto otro *agregando que antes no se había contemplado* . Por eso se dice que esta segunda etapa en el desarrollo del algoritmo es un *refinamiento* con respecto a la etapa anterior.

```
{2 etapa/refinamiento}
```

```
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café
 si ( está vacío el bote de café molido )
   moler café
fin si
agregar el café molido al contenedor
enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
```

Después hemos considerado que podríamos necesitar café cargado o ligero (en vez de poner siempre la misma cantidad de café). Llegamos a esta *tercera etapa* en la resolución del algoritmo (un *nuevo refinamiento*) que considera algo que antes no estaba contemplado

```
{3 etapa/refinamiento}
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café

si ( está vacío el bote de café molido )
    moler café
fin si

{     línea anulada  agregar el café molido al contenedor}
 si ( quiero el café cargado )
   agregar 4 cucharaditas de café
si no
   agregar 2 cucharaditas
fin si

enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
```

Finalmente, en la cuarta etapa del desarrollo del algoritmo hemos querido precisar en qué consiste eso de agregar n cucharadas de café. Esta vez, *no hay nada nuevo* pero lo que había lo *expresamos con más detalle*. También esto es otro *refinamiento*.

```
desenroscar los dos cuerpos de la cafetera
extraer el contenedor del café de la parte inferior
agregar agua hasta la válvula al cuerpo inferior
poner el contenedor de café

si (está vacío el bote del café molido)
   Moler café
fin si

si (quiero el café cargado)
    ctaCucharadas <-- 0
    mientras ctaCucharadas < 4
        poner una cucharada
        ctaCucharadas <-- ctaCucharadas + 1
    fin mientras
si no
    ctaCucharadas <-- 0
    mientras ctaCucharadas < 2
        poner una cucharada
        ctaCucharadas <-- ctaCucharadas + 1
    fin mientras
fin si

enroscar la cafetera
poner la cafetera al fuego hasta que salga el café
```

Resumiendo

La solución del algoritmo ha seguido una progresión por etapas

La primera etapa plantea una solución muy general, pero muy imprecisa.

Cada nueva etapa, se dice que es un refinamiento de la anterior porque representa

- agregar cosas no consideradas explícitamente antes o bien
- describir las cosas antes vistas con mayor detalle

Esta forma de progresar por etapas que cada vez consideran explícitamente más cosas y cada vez con más detalle se llama *diseño descendente por refinamientos sucesivos* usando *recursos abstractos*.

¿Qué es un recurso abstracto?

Considere la sentencia

```
agregar el café molido al contenedor
```

Se dice de una expresión como esta que es un recurso abstracto por dos razones.

Dice lo que se debe hacer pero no cómo se debe hacer. No considera todas las posibilidades que pueden presentarse o no las describe con detalle.

Sin embargo, en cuanto se le añaden los detalles precisos se convierte en un verdadero algoritmo (recordemos que el algoritmo

requiere precisión y considerar todas las posibilidades).
Por tanto, lejos de ser inútil, el recurso abstracto nos sirve de gran ayuda porque

da un punto de partida que con sucesivos refinamientos lleva a la solución

evita considerar desde la primera etapa TODA la complejidad del algoritmo lo que haría intratables la mayoría de los problemas.

algoritmo para construir algoritmos

1. *Construya una primera solución muy general que use recursos abstractos* .
2. *Haga evolucionar por etapas esa solución*
   1. *agregando cosas* no contempladas explícitamente, o bien
   2. *expresando con más detalle* lo que ya se ha contemplado antes
3. ¿Hasta donde debe progresar el desarrollo de cada recurso abstracto?
   1. Si el algoritmo se está escribiendo en el lenguaje de ordenador X, hasta que cada instrucción del algoritmo se pueda expresar en términos de una sentencia del lenguaje X.

Ejercicios

Un edificio rectangular tiene en su lado mayor una acera que se recorre en 100 pasos. Representar el recorrido de la acera sabiendo que estamos en uno de sus extremos. ver el código ocultar el código

Representar un recorrido por el perímetro de ese edificio rectangular suponiendo que el lado menor tiene 50 pasos y que se comienza en el centro del lado mayor. ver el código ocultar el código

Representar un recorrido por el perímetro de cualquier edificio rectangular cuyas medidas en pasos se piden al ejecutarse. El recorrido siempre empieza en una esquina. ver el código ocultar el código

Observaciones a los ejercicios

Posición inicial

- dicen Representar un recorrido ...
- no dicen donde estamos ahora
- por tanto, es válido suponer que estamos en el punto inicial

Dirección del recorrido

- El enunciado no indica si el recorrido se ha de hacer en el sentido de las agujas del reloj o al contrario. El sentido común tampoco.
- Por tanto, es tan razonable girar hacia la izquierda como hacia la derecha.

Si tienes una *duda* de interpretación

En la Universidad, *pregunta a tu profesor*

Cuando seas informático, *pregunta a tu cliente* .

Tres formas de repetir

Hay tres formas de expresar la repetición: *mientras* , *repetir* y *desde*

Mientras

Esta es la que hemos visto, se caracteriza por poner la CONDICIÓN AL PRINCIPIO. En consecuencia: si al principio la condición es falsa, nunca se ejecutarán las acciones que hay dentro de mientras.

```
ctaCucharadas <-- 0
mientras(ctaCucharadas < 4) {la condición al principio}
    poner una cucharada
    ctaCucharadas <-- ctaCucharadas + 1
fin mientras
```

Repetir

Pone LA CONDICIÓN AL FINAL. Las instrucciones/acciones se ejecutan SIEMPRE una vez por lo menos.

```
ctaCucharadas <-- 0
repetir
    poner una cucharada
    ctaCucharadas <-- ctaCucharadas + 1
 hasta  que (ctaCucharadas = 4)   { condición al final }
```

En estos dos casos es necesario que una acción ponga el contador de cucharadas con el valor inicial 0

Desde ... hasta

Ahora se representa de una forma más sintética

```
 desde ctaCucharadas <-- 1 hasta 4
    poner una cucharada
fin desde
```

Este código es equivalente a este otro

```
ctaCucharadas <-- 1
mientras (ctaCucharadas <= 4)
    poner una cucharada
    ctaCucharadas <-- ctaCucharadas + 1
fin mientras
```

Estas tres formas de controlar la repetición tienen

unas características que la definen

un contexto que las hace más apropiadas

- Siempre que haya que repetir usaremos una de estas construcciones
- Habrá que elegir cual de las tres según el contexto.

Variables

Hasta aquí hemos utilizado expresiones como calculadora, ctaCucharadas, pasosDados, ladoMayor, LadoMenor para referirnos a una memoria que contiene un dato.

El nombre que le hemos dado a esas memorias refleja su contenido. En adelante, vamos

VARIABLE: memoria, con un nombre que refleja su contenido.