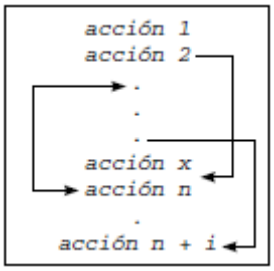
INTRODUCCIÓN

Normalmente los pasos de un algoritmo se ejecutan de arriba para abajo y se leen de izquierda a derecha. Pueden existir casos en los que sea necesario alterar esta secuencialidad por algún motivo. Generalmente este motivo radicará en que en cierto paso se evalúa una condición, y dependiendo del resultado de esa condición se optará por seguir un paso disponible. En esta sección veremos los tipos de operadores, cuyo resultado de evaluación determinan cual camino continuará el algoritmo. También, se introducirá el concepto de bifurcación condicional y su representación en un algoritmo.

BIFURCACION

Una bifurcación es una interrupción en el flujo normal de ejecución de un algoritmo, con lo cual la linealidad de ejecución se altera:



Las bifurcaciones pueden ser, según el punto del algoritmo donde se aplique de dos tipos: hacia adelante o hacia atrás (figura 1).

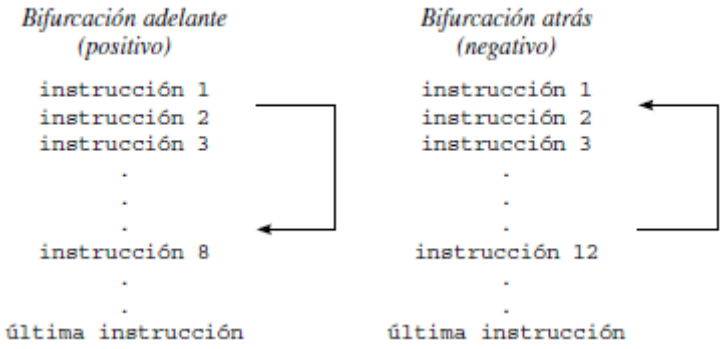


Figura 1. Tipos de bifurcaciones en un algoritmo

BIFURCACION CONDICIONAL Y LAS EXPRESIONES LOGICAS

Una bifurcación condicional es aquella que depende del cumplimiento de una determinada condición.

Observe el esquema de la figura 2. En él se puede visualizar la formalización de una evaluación de condición dentro de un rombo.

Si se cumple la condición se continuará el flujo de ejecución por la acción F2, en cambio si no se cumple la condición se ejecutará la acción F1. Una vez que se ejecuta la acción (o conjunto de acciones) señaladas se continuará con el flujo normal de ejecución, lo cual es mostrado cuando las fechas de la acción F1 y F2 que se unen en el círculo.

Como puede observar la idea central de la evaluación de la condición, es que devuelve si se cumple o no se cumple esta; y dado que el elemento central de almacenamiento de un valor dentro del algoritmo es la variable; se hace necesario que exista un tipo de datos capaz de almacenar los valores (Si y No, o dicho de otra manera VERDADERO o FALSO) que genera la evaluación de la condición.

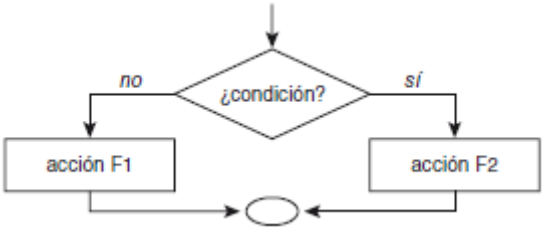


Figura 2. Bifurcación condicional

EL TIPO DE DATOS BOOLEAN (O LÓGICO)

Permite almacenar únicamente dos estados: verdadero o falso.

Una variable cuyo tipo de datos es Boolean se define de la siguiente manera:

variable: Boolean

Por ejemplo, una variable que cuyo valor indica si el jugador ha completado su misión dentro del juego se define de la siguiente manera

isMissionComplete: Boolean

O

haFinalizadoMision: Boolean

En general, se acostumbra a usar nombres en inglés para el identificador porque de alguna manera va a ser más corto y generalmente el verbo ser/estar estará presente, en cambio en su versión en español habrá que buscar un nombre adecuado a su función.

Como se ha comentado anteriormente este tipo de datos solo puede almacenar dos tipos de datos, que son verdadero y falso, así por ejemplo para asignarle el valor verdadero a la anterior variable se procedería de la siguiente manera:

isMissionComplete ← verdadero

Caso contrario lo realizaríamos de la siguiente manera

isMissionComplete ← falso

EXPRESIONES LÓGICAS

Todos los tipos de bifurcaciones que vamos a estudiar de aquí en adelante, en algún momento realizan la evaluación de una condición. Para que una condición devuelva un valor verdadero o falso se hace uso de expresiones lógicas.

En esencia, una expresión lógica (también denominada expresión booleana, en honor del matemático británico George Boole, que desarrolló el Álgebra de Boole) es una expresión que sólo puede tomar estos dos valores, verdadero o falso.

LOS OPERADORES RELACIONALES

Así como las expresiones aritméticas utilizan operadores aritméticos; **las expresiones lógicas utilizan operadores relaciones y lógicos**. A continuación, definiremos los operadores relacionales.

Los operadores relacionales permiten realizar comparaciones de valores de tipo numérico o carácter. Los operadores de relación sirven para expresar las condiciones en los algoritmos. Los operadores de relación se recogen en la Tabla 1. El formato general para las comparaciones es:

```
expresión1   operador de relación   expresión2
```

Y el resultado de la operación será verdadero o falso.

Tabla 1. Operadores relacionales

Operador	Significado
<	Menor que
>	Mayor que
= o ==	Igual que
<=	Menor o igual que
>=	Mayor o igual que
<> o !=	Distinto de

Veamos el efecto de estos operadores mediante el siguiente ejemplo:

Ejemplo: Suponga que posee dos variables denominadas numA y numB de tipo numéricas. El siguiente cuadro expresa los resultados que se obtienen al aplicar diversos operadores relacionales entre estas dos variables

numA	numB	Expresión lógica	Resultado
3	6	$numA < numB$	Verdadero
0	1	$numA > numB$	Falso
4	2	$numA = numB$	Falso
8	5	$numA <= numB$	Falso
9	9	$numA >= numB$	Verdadero
5	5	$numA <> numB$	Falso

Los operadores relacionales también se aplican a los tipos caracteres. El resultado de la comparación está determinado por reglas de ordenación de precedencia, definidas por un estándar denominado ASCII, que además de indicar la secuencia de ordenación de los caracteres de un alfabeto dentro del ordenador, también le asigna un valor numérico; que es el usado para realizar la ordenación de caracteres.

Si bien es cierto que no todos los ordenadores ni sistemas utilizan totalmente este estándar para algunos tipos de caracteres, en general se cumplirán las siguientes reglas:

- Los caracteres especiales #,%,$,+,-,/,…, exigen la consulta del código de ordenación

- Los valores de los caracteres que representan a los dígitos están en su orden natural. Esto es '0' < '1', '1' < '2', …
- Las letras mayúsculas A a Z siguen el orden alfabético.
- Si existen letras minúsculas, estas siguen el mismo criterio alfabético

Finalmente, si debemos comparar tipos de datos booleanos usando estos operadores siempre se cumplirá que

- verdadero > falso
- falso < verdadero

LOS OPERADORES LOGICOS

Estos operadores generalmente se van a utilizan para realizar combinaciones de expresiones lógicas que usan operadores relacionales. Los operadores y su significado están indicados en la tabla 2.

Tabla 2. Operadores lógicos

Operador Lógico	Expresión Lógica	Significado
no (not) o !	no p (not p)	Negación de p
Y (and) o &&	p Y q (p and q)	Conjunción de p y q
O (or) o ||	p O q (p or q)	Disyunción de p y q

Estos operadores trabajan sobre un resultado booleano. Es decir, evalúan una variable booleana o el resultado de una operación lógica basada en operadores relaciones y actúa en consecuencia según las denominadas tablas de verdad.

Para explicar estas tablas de verdad, suponga que a y b son variables booleanas o expresiones lógicas. Entonces las tablas de verdad de los operadores lógicos son y actúan de la siguiente manera:

Operador Lógico NO

a	no a
verdadero	falso
falso	verdadero

Ejemplo: sea a = (6 < 10) ¿Cuál es el resultado de aplicar **no** a?

Como puede observar si evaluamos la expresión lógica que almacena a, entonces verificaremos que la operación relacional (6 < 10) devuelve el valor verdadero.

Al aplicar el operador **no**, lo que hace es negar (asignar el valor opuesto) de a. Por lo tanto, el resultado de **no** a, será falso.

Operador Lógico Y (o conjunción)

El operador **Y** toma los valores que devuelven las expresiones lógicas a y b; y evalúa el resultado en conjunto.

Así, si tanto a como b son verdaderas, el resultado en conjunto será verdadero, caso contrario devolverá falso

a	b	a Y b
Verdadero	verdadero	verdadero
Verdadero	falso	falso
Falso	verdadero	falso
Falso	falso	falso

Ejemplo: sea a = (6 < 10) y b = (5 = 5) ¿Cuál es el resultado de aplicar a **Y** b?

- Si evaluamos a comprobaremos que su resultado es verdadero
- Si evaluamos b comprobaremos que su resultado es verdadero

Entonces (a **Y** b) deberá devolver un valor según lo que establece su tabla de verdad, que en este caso será verdadero. Observe que el operador de conjunción devuelve verdadero cuando ambos operandos (representados en la tabla como a y b) son verdaderos.

El Operador Lógico O (o disyunción)

a	b	a Y b
verdadero	verdadero	verdadero
verdadero	falso	verdadero
falso	verdadero	Verdadero
falso	falso	falso

El operador **O** toma los valores que devuelven las expresiones lógicas a y b; y evalúa el resultado en conjunto.

Así, el resultado será verdadero siempre y cuando al menos una de las expresiones relacionales o variables lógicas sea verdadera. Por tanto, la disyunción devuelve faso únicamente si tanto a como b son falsos.

Ejemplo: sea a = (6 < 10) y b = (5 = 5) ¿Cuál es el resultado de aplicar a **O** b?

- Si evaluamos a comprobaremos que su resultado es verdadero
- Si evaluamos b comprobaremos que su resultado es verdadero

Entonces (a **O** b) deberá devolver un valor según lo que establece la tabla de verdad. En este caso devolverá el valor verdadero.

DISEÑO DE UNA BIFURCACIÓN CONDICIONAL EN UN ALGORTIMO

A partir de ahora le daremos una mayor nomenclatura a la forma de escribir algoritmos. Esto se realiza debido a que es muy probable que los algoritmos que Ud. diseñe deban ser interpretados por otros diseñadores. Es decir, puede que Ud. diseñe un algoritmo y otro diseñador sea el que lo programe; o viceversa.

Esta situación requiere seguir diversas reglas sobre la manera de diseñar algoritmos; de tal manera que cualquier diseñador pueda interpretarlos. A esta situación se la conoce con el término de **Modelado**.

Existen diversas técnicas de modelado que se pueden utilizar. Normalmente, para quienes inician en el modelado de algoritmos que serán programados, una de las técnicas más utilizadas es el **Pseudocódigo**.

Un pseudocódigo está especialmente diseñado para dar énfasis a las instrucciones que representen cálculos matemáticos y evaluación de expresiones lógicas. De esta manera, logran que los programadores puedan compartir con otros programadores su visión de la resolución del problema.

Hay **dos principios en la escritura de un pseudocódigo**:

1. Al inicio se definen todas las variables que se usan en pseudocódigo; cada una con su nombre y su tipo (tal cual lo veníamos realizando hasta ahora)
2. Las líneas del pseudocódigo son órdenes (instrucciones o estructuras) que se ejecutan de arriba hacia abajo; primero una orden y después otra, así sucesivamente (tal cual lo hemos realizado hasta este momento).

La novedad se da en la forma en la que representan las bifurcaciones condicionales: el pseudocódigo puede contener de ser necesario **estructuras de control para romper con la secuencialidad y linealidad de las instrucciones**, las cuales se utilizan para describir las instrucciones de los algoritmos. Hay tres tipos de estructuras para este fin:

- Estructuras selectivas
- Estructuras iterativas
- Estructuras de anidamiento.

LAS ESTRUCTURAS DE CONTROL SELECTIVAS

Las estructuras selectivas permiten expresar las elecciones que se hacen durante la resolución del problema cuando existen un número de posibles alternativas a seguir como consecuencia de la evaluación de una condición.

Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas. En las estructuras selectivas se evalúa una condición y en función del resultado de esta se realiza una opción u otra.

Como se mencionó anteriormente, las condiciones se especifican usando expresiones lógicas.

Las estructuras selectivas o alternativas que ofrece un pseudocódigo son:

- Simples
- Dobles
- Múltiples

Estructura Selectiva Simple

La alternativa simple posee la siguiente estructura

Pseudocódigo en español	*Pseudocódigo en inglés*
si *<condición>* **entonces** *<acción Si>* **fin_si**	**if** *<condición>* **then** *<acción Si>* **endif**

Vamos a ver el uso de esta estructura en un ejemplo muy sencillo

Ejemplo: Leer dos números, si el primero es mayor que el segundo indicar como resultado el siguiente mensaje: "El primer número es mayor".

Fase de Análisis

- Especificación del Problema: Dado dos números determinar y mostrar si el primero es el mayor.
- Análisis:

  Datos de Entrada:

  numeroA, numeroB: Real

  Datos de Salida:

  mensajeResultante: String

  Proceso:

  //determinar al comparar dos números si el primero es mayor

  $numeroA > numeroB$ // devuelve un valor booleano

  // si se cumple lo anterior hay que mostrar un mensaje por la pantalla

  mensajeResultante = "El primer número es mayor"

Fase de Diseño

ENTIDAD QUE RESUEVE EL PROBLEMA: PersonaCalculando
VARIABLES numeroA, numeroB: real mensajeResultante: string
NOMBRE DEL ALGORITMO: comparar_numeros **PROCESO DEL ALGORITMO**: 1. *Leer* numeroA 2. *Leer* numeroB 3. **si** (numeroA > numeroB) **entonces** 4. mensajeResultante ← "El primer número es mayor" 5. *Mostrar* mensajeResultante 6. **fin_si**

Se podría obviar la numeración de cada paso, ya que en el pseudocódigo se asume la estructura secuencia de arriba para abajo y de izquierda a derecha. Al inicio se recomienda numerar para facilitar el seguimiento de los pasos cuando necesitan detectar errores.

Observe que luego de leer los valores para `numeroA` y `numeroB`, se invoca una estructura selectiva simple. Esta estructura selectiva simple se denomina **`si`** (if en inglés). Una parte importante de la estructura de control **`si`** es su clara delimitación del inicio y fin de su accionar; el cual queda establecido por el **`fin_si`** (endif).

Otro aspecto importante es que luego del **`si`** se evalúa una expresión lógica. En este ejemplo, a través del operador relacional > estamos evaluando si el valor asignado a `numeroA` es mayor

que el asignado a `numeroB`. Únicamente en el caso de que esta expresión lógica devuelva verdadero, se mostrará el mensaje "El primer número es mayor"

¿En caso de que no se cumpla la condición que sucede? Simplemente se ignoran las instrucciones dentro de la estructura selectiva delimitada por **`si`** … **`fin_si`**.

Estructura Selectiva Doble

La alternativa doble posee la siguiente estructura

Pseudocódigo en español	Pseudocódigo en inglés

`si <condicion> entonces`
`  <accion S1>`
`si_no`
`  <accion S2>`
`fin_si`
	`if <condicion> then`
	`  <accion S1>`
	`else`
	`  <accion S2>`
	`endif`

Vamos a ver el uso de esta estructura en un ejemplo muy sencillo

Ejemplo: Leer un número, si el número es par mostrar el mensaje "Es par", caso contrario mostrar el mensaje "Es impar".

Fase de Análisis

- Especificación del Problema: Dado un número determinar y mostrar si es impar o par.
- Análisis:

  Datos de Entrada:

  numeroIngresado: Entero

  Datos de Salida:

  mensajeResultante: String

  Proceso:

  //el módulo representa el resto de una división entre dos números. Si se divide un número por 2 y el resto vale 0, entonces estamos ante un número par

  $(numeroIngresado \ \% \ 2) = 0$ // devuelve un valor booleano

  // si se cumple lo anterior hay que mostrar un mensaje por la pantalla

  mensajeResultante = "Es Par"

  // caso contrario el mensaje será

  mensajeResultante = "Es Impar"

Fase de Diseño

ENTIDAD QUE RESUEVE EL PROBLEMA: PersonaEvaluandoNumero
VARIABLES numeroIngresado: Entero mensajeResultante: string
NOMBRE DEL ALGORITMO: evaluar_tipo_paridad **PROCESO DEL ALGORITMO**:

```
1. Leer numeroIngresado
2. si ((numeroIngresado % 2) = 0) entonces
3.         mensajeResultante ← "Es par"
4. si_no
5.         mensajeResultante ← "Es impar"
6. fin_si
7. Mostrar mensajeResultante
```

Nuevamente se utiliza una expresión lógica para evaluar la condición. Observe que se realiza en primer lugar una expresión aritmética (**`numero % 2`**). Esta expresión aritmética devuelve un valor, que puede ser cero u otro valor. Dado que por definición un número es par si su módulo al dividir por 2 es cero; se pregunta si el módulo es igual a cero. En este caso asigna a la variable `mensajeResultante` el valor "Es par". Pero si no se cumple la condición (si da como resultado falso) se ejecutará las instrucciones indicadas luego del si_no.

Estructura Selectiva Múltiple o Estructura SEGUN

Con mucha frecuencia cuando se evalúa una condición lógica puede presentarse la necesidad de que se compare o relacione la expresión contra un conjunto de diferentes posibles valores. Para poder diseñar un algoritmo que pueda cumplir con este tipo de planteamiento se debería hacer uso de los denominados **Si anidados**, de tal manera que se pueda contemplar la totalidad de las variantes que puede adoptar una variable como consecuencia de la evaluación de condición. Es decir, en estas situaciones se puede recurrir a estructuras alternativas simples o dobles, anidadas o en cascada.

Sin embargo, en este método si el número de alternativas es grande se puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad; y la complejidad en la lectura del algoritmo puede llevar al desarrollador a incurrir en errores. La siguiente estructura permite evitar este tipo de situaciones.

La estructura SEGÚN se define como una estructura de decisión múltiple, ya que permite evaluar una expresión que podrá tomar **`n`** valores distintos, 1, 2, 3, 4, ..., n.

Según se elija uno de estos valores en la condición, se realizará una de las **`n`** acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los **`n`** posibles.

La forma de esta estructura es la siguiente:

```
según_sea (expresión) hacer
   caso expresión constante :
      [Sentencia
       sentencia
       ...
       sentencia de ruptura | sentencia ir_a ]
   caso expresión constante :
      [Sentencia
       sentencia
       ...
       sentencia de ruptura | sentencia ir_a ]
   caso expresión constante :
      [Sentencia
       ...
       sentencia
       sentencia de ruptura | sentencia ir_a ]
   [otros:
      [Sentencia
       ...
       sentencia
       sentencia de ruptura | sentencia ir_a ]
fin_según
```

Cabe recalcar que para este tipo particular de estructura de control existen diversos modelos de representación en pseudocódigo, siendo todos ellos válidos. Sin embargo, en esta asignatura adoptaremos el modelo previamente esquematizado.

Ejemplo: Se desea diseñar un algoritmo que escriba los nombres de los días de la semana en función del valor de una variable **día** introducida por teclado.

Fase de Análisis

- Especificación del Problema: Mostrar el día de la semana a partir de un número que representa el día de la semana.
- Análisis:

  Datos de Entrada:

  dia: Entero

  Datos de Salida:

  mensaje: String

  Proceso:

  //Se utiliza una correspondencia entre valores numéricos y el texto de un día

  // Así 1=Lunes, 2=Martes, 3= Miércoles, 4=Jueves, 5=Viernes, 6=Sábado, 7=Domingo

  // si se cumple lo anterior hay que mostrar el texto del día de la semana

  // caso contrario el mensaje será

  mensaje = “Error”

Fase de Diseño

NOMBRE DE LA ENTIDAD QUE RESUELVE EL PROBLEMA: Calendario
VARIABLES: dia: Entero // representa numéricamente el día de la semana ingresado mensaje: string // representa el mensaje que se mostrará (el día en forma de texto)
NOMBRE DEL ALGORITMO: mostrar_dia_de_semana **PROCESO DEL ALGORITMO**: 1. *Leer* día 2. **según_sea** (día) hacer 3. **caso** 1: 4. mensaje ← “LUNES" 5. sentencia de ruptura 6. **caso** 2**:** 7. mensaje ← “MARTES" 8. sentencia de ruptura 9. **caso** 3**:** 10. mensaje ← “MIERCOLES” 11. sentencia de ruptura

```
12.     caso 4:
13.         mensaje ← "JUEVES"
14.         sentencia de ruptura
15.     caso 5:
16.         mensaje ← "VIERNES"
17.         sentencia de ruptura
18.     caso 6:
19.         mensaje ← "SABADO"
20.         sentencia de ruptura
21.     caso 7:
22.         mensaje ← "DOMINGO"
23.         sentencia de ruptura
24.     otros:
25.         mensaje ← "ERROR"
26.         sentencia de ruptura
27. fin_segun
28. Escribir mensaje
```

En este caso puede observar que la estructura **Según** es de las más costosas desde el punto de vista computacional, ya que la variable en cuestión es comparada con cada una de las opciones disponibles para el caso en el cual no es igual a ninguna de las alternativas.

Uno de los usos más comunes que se le dan a esta estructura selectiva es la realización de menús de opciones.

BIBLIOGRAFIA

Fundamentos de Programación: Algoritmos, estructuras de datos y objetos. Cuarta edición. Luis Goyanes Aguilar. ISBN: 978-84-481-6111-8