Metodología de la Programación

Autor: Xavi Llunell

[Ver curso online]

Presentación del curso

Este curso gratis le proporcionará, a modo de iniciación, algunos de los pasos a seguir para aprender a programar.Los bucles o los operadores lógicos son algunos de los contenidos que aprenderá en este curso. Un curso realizado en colaboración con Areaint.com.

Visita más cursos como este en mailxmail:
[http://www.mailxmail.com/cursos-informatica]
[http://www.mailxmail.com/cursos-programacion]

¡Tu opinión cuenta! Lee todas las opiniones de este curso y déjanos la tuya:
[http://www.mailxmail.com/curso-metodologia-programacion/opiniones]

Cursos similares

Cursos	Valoración	Alumnos	Vídeo
PHP y MySQL. Aplicaciones Web (undécima parte) Programación de aplicaciones Web con PHP y MySQL. Ahora te capacitamos para entender el funcionamiento en Internet de una tienda online. Aprenderás ... [02/12/08]		1.367	
Funciones en C (primera parte) Curso de programacíon informática sobre Funciones en C en el orden de los fundamentos de la programación. Comprende el desarrollo de un software utilizando funciones y a ... [17/10/08]		5.595	
Manual de programación El objetivo de este curso consiste en ofrecer conocimientos básicos de programación. No pretender enseñar cómo programar en un lenguaje específico, ni utilizar ninguna te... [31/01/08]		2.801	
Holística informática Aprende con nuestro curso de holística informática, sobre los avances informáticos que te ayudará a actualizarte conociendo más acer... [28/04/09]		2.223	
Primeros pasos con XML y XSL XML es el acrónimo del inglés eXtensible Markup Language cuyo objetivo principal es conseguir una página web más semántica. Inicialmente nace como sucesor del HTML, separ... [10/09/04]		7.410	

1. Introducción

[http://www.mailxmail.com/curso-metodologia-programacion/introduccion]

En este curso veremos las estructuras básicas de la programación. Estas estructuras nos ayudarán en el momento en el que nos tengamos que enfrentar a la creación de un programa, ya que nos facilitarán la planificación y la programación del mismo.

En el momento de programar deberá tener en cuenta los siguientes aspectos.

-Estructurar el código para una fácil comprensión en el momento de modificaciones y ampliaciones.

-Poner comentarios en lugares clave para facilitar el entendimiento del código.

-Intentar ahorrar el número de líneas de código, cuantas más líneas inútiles peor será el programa.

-Pensar que el código del programa, muchas veces, no es sólo para nosotros, sino que hay mucha gente a la que le puede interesar nuestro trabajo o deban trabajar con él y ellos deben ser capaces de entender el código.

El por qué de la Metodología.- Mucha gente piensa que estudiar metodología de la programación es una cosa ardua, costosa y muy aburrida. Nosotros intentaremos hacer que esto no sea así y que todo lo que aprenda a partir de este momento le sea de mucha utilidad para la creación de esos programas que tiene en mente.

Ejemplo (Subiendo escaleras).- Imagine que estamos creando un pequeño programa para un robot que debe subir 10 escalones. El robot entiende las siguientes instrucciones **LevantePieIzquierdo** (para levantar el pie izquierdo y subir un escalón) y **LevantePieDerecho** (para levantar el pie derecho y subir otro escalón), con lo que podrá ir ascendiendo hasta llegar al final de una escalera.

Si sólo pudiésemos utilizar estas dos instrucciones, para crear un programa para que el robot subiera los 10 escalones, deberíamos hacer un programa con las siguientes líneas de código:

LevantePieIzquierdo
LevantePieDerecho
LevantePieIzquierdo
LevantePieDerecho
LevantePieIzquierdo
LevantePieDerecho
LevantePieIzquierdo
LevantePieDerecho
LevantePieIzquierdo
LevantePieDerecho

Fíjese que en este caso. Hemos tenido que escribir las mismas instrucciones varias

veces para que el robot subiera los escalones. ¿Qué hubiese sucedido si el robot en lugar de subir 10 escalones hubiese tenido que subir los 1.665 escalones de la Torre Eiffel? El código hubiese sido interminable, corriendo el peligro de equivocarnos al contar la cantidad de escalones, con lo que el robot no hubiese llegado a la cima de la torre. O incluso, nos podríamos haber equivocado escribiendo dos veces la misma instrucción, con lo que el robot se hubiese tropezado al levantar dos veces el mismo pie.

2. Formas de solucionar posibles problemas

[http://www.mailxmail.com/...rso-metodologia-programacion/formas-solucionar-posibles-problemas]

Para solucionar posibles problemas de repetición de la misma instrucción o de códigos interminables, disponemos de diferentes instrucciones que nos permiten reducir el número de líneas de un programa facilitando así la compresión, la modificación del código y un posible error en la ejecución del programa.

Observe una posible solución para nuestro problema. (Piense que para un mismo problema no sólo existe una solución, yo le ofreceré una, pero eso no quiere decir que sea la mejor). Las instrucciones que aparecen en esta solución se irán explicando a lo largo del curso.

```
Repetir hasta que NúmeroEscalón = 5
    LevantePieIzquierdo
    LevantePieDerecho
    Sume 1 a NúmeroEscalón
Fin Repetir
```

Lo que hemos creado es una repetición de código hasta que se cumple una determinada condición. Compare las dos posibles soluciones al mismo programa.

Seguramente, al no conocer las instrucciones de la segunda solución, le parecerá mucho más fácil la primera solución, pero estoy seguro que cuando haya realizado alguna lección más del curso comprenderá que la segunda es mucho más fácil de comprender y de modificar que la primera. Ya intentaré demostrárselo a lo largo de todo el curso.

3. La estructura condicional 'si'

[http://www.mailxmail.com/curso-metodologia-programacion/estructura-condicional-si]

En este e-mail nos adentraremos en el concepto de condición.

Una instrucción condicional es aquella que nos permite "preguntar" sobre el entorno que nos rodea, pudiendo así actuar según la respuesta obtenida. Estas respuestas siempre serán Verdaderas o Falsas, pudiendo así tomar, en un principio, dos caminos diferentes.

Pongamos un pequeño ejemplo. Imagine que tenemos nuestro robot subiendo las escaleras de la Torre Eiffel y le indicamos lo siguiente: si está en el escalón 200 entonces no continúe subiendo, sino continua su ascensión. Como puede ver aquí se nos presenta una condición y, según en el punto de la escalera en la que se encuentre, nuestro robot se parará o continuará subiendo.

Estructura básica de una condición.- En nuestro curso esta estructura condicional se representará de la siguiente manera:

Si <CONDICIÓN> entonces
<INSTRUCCIÓN 1>
Si no
<INSTRUCCIÓN 2>
Fin Si

Le explicaremos línea a línea la estructura condicional anterior:

-Primera línea: aparece parte de la estructura condicional y la **<CONDICIÓN>**que deseamos evaluar.

-Segunda línea: lugar donde pondremos las instrucciones a realizar en caso que la respuesta a la condición sea **VERDADERA**.

-Tercera línea: parte de la estructura condicional que nos divide la parte de las instrucciones a realizar cuando la condición es verdadera y cuando es falsa.

-Cuarta línea: lugar donde pondremos las instrucciones a realizar en caso que la respuesta a la condición sea **FALSA**.

-Quinta línea: parte de la estructura condicional que nos indica el final de la misma.

En la siguiente unidad didáctica le explicaremos un ejemplo.

4. Ejemplo 1 - subiendo escaleras hasta el escalón 200

[http://www.mailxmail.com/...metodologia-programacion/ejemplo-1-subiendo-escaleras-escalon-200

Ahora veremos cómo podemos utilizar esta estructura condicional dentro del caso que hemos citado anteriormente.

Recordemos el problema: tenemos nuestro pequeño Robot que sube una escalera. En un momento determinado, queremos que tome una decisión según el punto en el que se encuentra. La decisión es la siguiente: si se encuentra en el escalón 200 debe detenerse, mientras que si está en cualquier otro sitio debe continuar subiendo.

Ante un problema así, debemos plantearnos cuál es la estructura que debemos utilizar. En este caso, es fácil ya que sólo hemos estudiado una. Cuando tenemos clara la estructura debemos mirar que es lo que debemos poner en cada lugar y cuáles son las instrucciones que debemos utilizar.

Vamos a definir las instrucciones que nos hacen falta para este ejemplo:

-Escalón: nos servirá para saber en qué número de escalón se encuentra nuestro robot.

- **Parar:** detiene al robot y lo pone en modo de pausa para que "descanse".

Estas "instrucciones" están hechas a medida porque estamos utilizando un pseudo código, pero podrían ser sustituidas por instrucciones reales en el momento en el que nosotros decidiéramos en qué lenguaje realizar nuestro programa.

Una vez hemos definido las instrucciones y mirado cuál es la estructura que queremos utilizar pasaremos a escribir el código. Quiero decir que al principio puede ser que le cueste un poco encontrar una solución lógica. Pero piense que esto, como la mayoría de las cosas, requiere mucha paciencia, mucha práctica y algo de concentración.

Pasemos a implementar el ejemplo:

```
Si Escalón = 200 entonces
    Parar
Sino
    LevantaPieIzquierdo
    LevantaPieDerecho
Fin Si
```

Observe detenidamente la implementación del código y mire cuáles serían los pasos que deberíamos seguir cuando se dieran los dos posibles casos de nuestro problema. Recuerde que para un mismo problema pueden existir diferentes soluciones.

5. Ejemplo 2 - Moverse de una posición a otra

[http://www.mailxmail.com/curso-metodologia-programacion/ejemplo-2-moverse-posicion]

Una vez visto el ejemplo anterior vamos a plantear otros que nos ayudarán a entender y a familiarizarnos con esta estructura condicional. Con la práctica, podrá ver que esta es una de las estructuras que más se utilizan en programación. En casi todos los programas hay que tomar algún tipo de decisión.

Este es un ejemplo un poco tonto pero nos puede servir para acabar de entender cómo funciona esta estructura condicional.

Imagine que tenemos a nuestro Robot trabajando en una cadena de montaje en una plataforma móvil que tiene dos posiciones posibles: la 1 ó la 2 a derecha e izquierda.

Lo que nosotros queremos que haga el Robot es lo siguiente: si el Robot inicialmente se encuentra en la posición 1 deberá moverse a la 2, y si se encuentra en la posición 2 deberá moverse a la 1.

Para este ejemplo definiremos unas nuevas instrucciones que iremos utilizando en futuros ejemplos:

-MoverDer: moverá el robot a la derecha una sola posición.

-MoverIzq: moverá el robot a la izquierda una sola posición.

-Posición: esta instrucción nos servirá para saber en qué posición se encuentra el robot.

Antes de mirar la solución que yo le propongo estudie una posible solución, piense un poco y no tenga miedo en equivocarse con la solución.

Posible solución Ejemplo 2.- Una posible solución a nuestro programa del Robot que se debe mover de una posición a la otra podría ser la siguiente:

```
Si Posición = 1 entonces
   MoverDer
Sino
   MoverIzq
Fin Si
```

Observe que en este ejemplo lo que hacemos es preguntar si el Robot se encuentra en la posición 1. Si la respuesta es Verdadera haremos que se mueva a la derecha **"MoverDer"** (con lo que se quedará en la posición 2), mientras que si la respuesta fuera Falsa (cosa que querría decir que está en la posición 2) se movería a la izquierda **"MoverIzq"** (quedando situado el Robot en la posición 1). Así de fácil podemos hacer que nuestro Robot cambie de posición dependiendo de su posición inicial.

6. Ejemplo 3 - Coger el bloque y moverse

[http://www.mailxmail.com/...curso-metodologia-programacion/ejemplo-3-coger-bloque-moverse]

Ahora que ya sabemos cómo hacer que nuestro Robot se mueva según la posición en la que se encuentra; vamos ha realizar un ejercicio que nos haga lo siguiente: tenemos a nuestro Robot en la plataforma móvil definida anteriormente, pero esta vez con tres posiciones: 1 - 2 - 3. Y encima de una de estas tres posiciones pondremos un bloque (caja) llamada **"A"**. Puede que en alguna de estas posiciones haya un bloque que no se llame **"A"**, si fuera así no deberíamos hacer nada con él.

Lo que queremos que realice nuestro Robot es lo siguiente: el Robot siempre partirá de la posición 1, pero no sabrá dónde está situado el bloque y éste siempre debe estar en una de las 3 posiciones. Nosotros deberemos hacer que nuestro Robot se desplace por la cinta mirando si el bloque se encuentra en esta posición. Si el bloque está en esa posición, debe cogerlo.

Como siempre, vamos a definir las nuevas instrucciones que necesitaremos para solucionar este problema, recuerde que si necesita alguna de las instrucciones definidas anteriormente puede hacerlo. Eso sí, no puede inventarse ninguna instrucción nueva. Es recomendable que vaya haciendo una lista de todas las instrucciones que vamos definiendo.

Nuevas instrucciones:

- **BloqueEncima:** será el equivalente a hacer que el robot mire arriba y mire el nombre que tiene escrito el bloque.

-CogerBloque: el robot alargará su brazo y cogerá el bloque.

Sería muy recomendable que antes de mirar la solución lo intentara usted solo, para ver si ha comprendido a la perfección todo lo que llevamos de curso.

7. Posible solución al ejemplo 3

[http://www.mailxmail.com/curso-metodologia-programacion/posible-solucion-ejemplo-3-1

Vamos a dar dos posibles soluciones para el problema que se nos plantea, en este e-mail y en el siguiente. Recordemos que el Robot siempre partirá de la posición 1 tal como indica el enunciado.

Primera solución.-

```
1.- Si BloqueEncima = "A" entonces
2.-    CogerBloque
3.- Sino
4.-    MoverDer
5.-    Si BloqueEncima = "A" entonces
6.-       CogerBloque
7.-    Sino
8.-       MoverDer
9.-       CogerBloque
10.-   Fin Si
11.- Fin Si
```

Explicación del ejemplo.-

1.- Miramos si encima de la primera posición, posición en la que sabemos que se encuentra seguro al iniciar el programa, está el bloque A.

2.- Si es así, **<CONDICIÓN>** verdadera lo cogemos. En este caso, hemos terminado el programa. Ya que continuaría con la línea 3, punto donde se separa la parte de la respuesta a la condición verdadera y falsa, y al haberse cumplido la parte Verdadera saltaríamos hasta la línea 11.

3.- Si no se cumple la **<CONDICIÓN>** puesta en la línea 1, continuamos el programa.

4.- Hacemos desplazar al Robot hacia la derecha. Hecho esto, nos encontramos en la posición 2 de nuestra plataforma.

5.- Miramos si encima de esta posición (segunda posición) está el bloque A.

6.- Si es así, **<CONDICIÓN>** verdadera la cogemos. En este caso, hemos terminado el programa; ya que continuaría con la línea 7, punto donde se separa la parte de la respuesta a la condición verdadera y falsa, y al haberse cumplido la parte Verdadera saltaríamos hasta la línea 10.

7.- Si no se cumple la **<CONDICIÓN>** puesta en la línea 5, continuamos el programa.

8.- Hacemos desplazarse el Robot a una posición derecha. Hecho esto, nos encontramos en la posición 3 y última de nuestra plataforma.

9.- Como sabemos seguro que el bloque "A" está en alguna de las 3 posiciones y ya hemos visto que no estaba ni en la 1, ni en la 2. Sin mirar qué bloque hay en esta posición podemos cogerlo.

10.- Fin del segundo **Si.**

11.- Fin del primer **Si**.

Observe como dentro de un **Si** hemos puesto otro **Si**. Con lo que hemos podido hacer una pregunta después de haber visto que la primera era falsa.

Observe que para este ejemplo hemos necesitado 11 líneas y muchas de estas líneas están repetidas varias veces. Esto no quiere decir que no funcionaría, ya que hemos visto en la traza, ejecución del programa línea a línea, que el Robot al final conseguiría su objetivo. Cuando veamos que una misma instrucción se repite muchas veces nos deberíamos plantear que puede ser que exista alguna otra solución que sea un poco más corta y con menos líneas repetidas.

8. Otra posible solución al ejemplo 3

[http://www.mailxmail.com/curso-metodologia-programacion/posible-solucion-ejemplo-3-2

Ahora veremos una solución al mismo problema en el que sólo se utilizan 7 líneas.

```
1.- Si BloqueEncima <> "A" entonces
2.-     MoverDer
3.-     Si BloqueEncima <> "A" entonces
4.-         MoverDer
5.-     Fin Si
6.- Fin Si
7.- CogerBloque
```

1.- Preguntamos si el posible bloque que hay en esta posición es diferente a **"A".** De esta forma, si el bloque fuera el **"A"**, saldríamos del bucle en la línea 6.

2.- Nos moveríamos una posición a la derecha, con lo que ya estaríamos situados en la segunda posición.

3.- Preguntamos si el posible bloque que hay en esta posición es diferente a **"A".**

4.- Si estamos en esta línea quiere decir que el bloque **"A"** no estaba en la primera posición, ni en la segunda; con lo que nos desplazaremos otra posición a la derecha para situarnos en la tercera posición.

5.- Línea que nos indica el final del segundo **Si.**

6.- Línea que nos indica el final del primer **Si.**

7.- En esta línea, sería el momento en el que cogemos el bloque que existe en la posición que nos encontramos.

Si se fija en este ejemplo, la instrucción **CogerBloque** está fuera del **Si** ya que, lo que hemos conseguido, con todas las instrucciones que tenemos dentro de nuestra estructura condicional, ha sido situar el Robot en la posición en la que se encuentra el bloque **A**.

Observe que, en este caso, desde un principio preguntamos por diferente **(<>)** y no por igual **(=);** con lo que conseguimos que la forma de plantearnos el código sea un poco diferente.

Aquí le muestro una lista de operadores lógicos que podemos utilizar:

- **=** Igual que
- **<** Menor que
- **>** Mayor que
- **< >** Diferente que

Intente ampliar los ejemplo que hemos dado hasta este momento y buscar una implementación del código lo más sencilla posible. En próximas lecciones, pondremos más ejemplos y algunos ejercicios a realizar.

9. Los bucles

[http://www.mailxmail.com/curso-metodologia-programacion/bucles-1]

Un bucle no es más que una serie de instrucciones que se repiten.

Podemos tener dos tipos de bucles según lo que nos interese comprobar.

- Bucle que se repite mientras se cumple una condición determinada.

- Bucle que se realiza hasta que se cumple la condición que marcamos.

En esta lección nos encargaremos del primer tipo.

A la hora de utilizar un bucle, sea del tipo que sea, debemos ir con cuidado y pensar cuando debe acabar ya que; si no tuviéramos en cuenta esto, podríamos entrar en un bucle sin fin, bucle que no terminaría nunca. Por esto, es de suma importancia que pensemos, antes de hacer nada, en qué momento, cómo, dónde y por qué debe acabar el bucle.

Estructura básica.- En nuestro curso, es la estructura básica de un bucle y se representará de la siguiente manera:

Mientras <CONDICIÓN> hacer
    <INSTRUCCIÓN / Instrucciones>
Fin Mientras

Explicaremos línea a línea este bucle:

-Primera línea: en esta línea escribiremos la **<CONDICIÓN>** que se debe dar para que se ejecute el bucle. En el momento en que la condición que estamos evaluando sea Falsa, se terminará el bucle y se continuarán ejecutando las instrucciones que aparecerían después de la tercera línea de esta estructura básica.

-Segunda línea: línea o líneas donde escribiremos las instrucciones a efectuar en caso de que la condición sea Verdadera.

-Tercera línea: marca el final del bucle. Podemos decir que nos indica hasta dónde están las líneas que se repiten o justo el final del bucle para poder seguir con las demás instrucciones de nuestro programa.

En esta estructura no tenemos ninguna línea que sea opcional.

10. Ejemplo 1 - Situarse en la última posición de nuestra plataforma (I)

[http://www.mailxmail.com/...gramacion/ejemplo-1-situarse-ultima-posicion-nuestra-plataforma-1]

Imagine que tenemos a nuestro Robot en la plataforma móvil que definimos en lecciones anteriores, con tres posibles posiciones 1 - 2 - 3.

Nuestro Robot lo podemos tener situado en la primera o segunda posición y nosotros queremos que se desplace hasta la tercera posición.

Pasemos a implementar el ejemplo:

```
Mientras Posición <> 3 hacer
    MoverDer
Fin Mientras
```

Observe el código que hemos propuesto para la resolución de este primer ejemplo con bucles. Vamos a seguir paso a paso cómo miraríamos si hemos realizado bien la solución del código en los dos posibles casos (Robot posicionado en la posición 1 ó en la 2).

-Tenemos el Robot en la posición 1

a)Miraríamos la primera línea, la que marca la condición. La condición nos dice que entremos en el bucle mientras no estemos en la posición 3 (**Mientras Posición <> 3 hacer**). Como en este caso no estamos en la posición indicada, ya sabemos lo que nos toca, entrar en el bucle.

b)Nos movemos una posición a la derecha **(MoverDer).**

c)Vamos a la última línea del bucle. **(Fin Mientras).**

d)Volvemos a mirar si se cumple la condición. Como todavía no se cumple, ya que nos encontramos en la posición 2, entraríamos otra vez dentro del bucle.

e)Nos volveríamos a desplazar una posición a la derecha.

f)Volveríamos al final del bucle donde nos mandaría otra vez a la primera línea de éste para así volver a comprobar la condición.

g)Ahora, como ya nos encontramos en la posición deseada, no entraríamos dentro de nuestro bucle y seguiremos con la instrucción que encontrásemos después de (**Fin Mientras**); en este caso finalizamos el programa.

11. Ejemplo 1 - Situarse en la última posición de nuestra plataforma (II)

[http://www.mailxmail.com/...gramacion/ejemplo-1-situarse-ultima-posicion-nuestra-plataforma-2]

Continuamos con el ejemplo que planteamos en la lección anterior.

-Tenemos el Robot en la posición 2.

a)Miramos la condición que nos pone la estructura **Mientras**, como no nos encontramos en la posición 3, entramos en el bucle.

b)Nos movemos una posición hacia la derecha (**MoverDer**).

c)Vamos a la última línea del bucle. (**Fin Mientras**), la cual nos manda a la primera línea del bucle.

d)Volvemos a mirar si se cumple la condición. Vemos que ya nos encontramos en la tercera posición. Ya no hace falta que entremos en el bucle. Ya podemos seguir con las instrucciones que tenemos después del bucle.

A este proceso que acabamos de realizar le llamaremos **Traza**. **La traza, realizar paso por paso un código, es muy importante en el momento de poder detectar posibles errores en nuestra implementación del código; ya que nos podemos dar cuenta si nuestro código falla o realiza alguna acción que no deseamos.** Es muy importante que al realizar la traza hagamos todos y cada uno de los pasos indicados y no demos nada por supuesto. En el momento de realizar la traza debemos actuar como si nosotros fuéramos la máquina que realizará el código.

12. Ejemplo 2 - Bucle infinito

[http://www.mailxmail.com/curso-metodologia-programacion/ejemplo-2-bucle-infinito]

Vamos a ver un pequeño ejemplo en el que el bucle sería infinito, con lo que nos podría provocar un error en el momento de ejecutar el programa.

Imagine que tenemos el siguiente código. (Este código no tiene enunciado)

Mientras Posición > 1 hacer
MoverDer
Fin Mientras

Imagine que nuestro Robot, en un principio, estuviera situado en la posición 2, ¿qué pasaría?.

Pues muy sencillo, que comenzaría a desplazarse hacia la derecha y como a partir de este momento todas las posiciones son mayores de 1, él siempre se movería una posición a la derecha con el peligro de salirse del raíl en el que está trabajando, produciendo así un **ERROR**. Si realizáramos la traza correspondiente a nuestro código veríamos cuál es el error y cómo lo podríamos solucionar.

Para realizar la traza imagine que estamos trabajando en un raíl con 3 posiciones y que partimos de la primera posición así, además, se acostumbrará a ver cómo funciona el bucle.

Ejemplo 3.Bloque sobre la plataforma.- En este ejemplo, vamos a utilizar las dos estructuras que hemos visto hasta este momento.

El enunciado sería el siguiente: tenemos, otra vez, a nuestro Robot montado en la plataforma móvil pero un poco ampliada ya que, esta vez, tenemos 5 posiciones. Nosotros, inicialmente, podemos encontrar el Robot en una de las 4 posiciones iniciales y un bloque llamado **A** en una de ellas. (La colocación del Robot y del bloque será aleatoria). El bloque siempre estará situado en alguna posición. Nosotros queremos encontrar ese bloque, esté donde esté, y llevarlo hasta la quinta posición que siempre estará vacía.

Antes de mirar la solución que le propongo intente encontrar por si sólo una solución correcta. Una vez la tenga realice una traza para ver si los pasos que ha pensado realizan lo que le hemos planteado. Consulte, en todo momento, la lista de instrucciones que se ha construido hasta este momento.

13. Posible solución al ejemplo 3

[http://www.mailxmail.com/curso-metodologia-programacion/posible-solucion-ejemplo-3-3

Antes de realizar una implementación de nuestro problema, debemos tener en cuenta algo que puede ser que se nos haya pasado por alto. Observe que en el enunciado del problema se nos dice que el Robot puede estar en cualquiera de las 4 primeras posiciones al igual que el bloque **A**. Por lo tanto, puede ser que el bloque quede colocado en la posición 1 y el Robot en la 2. Con lo que tendremos que pensar alguna manera para hacer que éste, estando en cualquier posición, pueda recorrer todas las posiciones mirando si en ellas está el bloque **A**.

Pasemos a la implementación del código y a su correspondiente explicación. Esta no es la única manera que tenemos para realizar la solución.

Enumeraremos las líneas para poder hacer una explicación del funcionamiento de nuestro código.

```
1.- Mientras Posición <> 1 hacer
2.-       MoverIzq
3.- Fin Mientras
4.- Mientras Posición <> 5 hacer
5.-     Si BloqueEncima = "A" entonces
6.-          CogerBloque
7.-     Fin Si
8.-     MoverDer
9.- Fin Mientras
```

En la próxima lección le explicaremos cada una de las líneas de este ejemplo.

14. Explicación al ejemplo anterior

[http://www.mailxmail.com/...curso-metodologia-programacion/explicacion-ejemplo-anterior-1]

A continuación le explicamos una a una las líneas del ejercicio de la lección anterior.

1.- Miramos si la posición del Robot es diferente a 1. De esta forma, conseguiremos, estando en la posición que estemos, situarnos en la primera posición para tratar siempre el problema de la misma forma. Esta ¿preparación del terreno¿ es muy usual dentro de la programación ya que nos ayuda a plantearnos todos los casos como un caso único.

2.- Nuestro Robot se moverá una posición a la izquierda.

3.- Indique el final del bucle y nos posicionará nuevamente en la línea 1 para volver a mirar si se cumple o no la condición.

Después de asegurarnos que nuestro Robot se encuentra en la posición 1. Iniciamos lo que sería la resolución del problema planteado.

4.- Inicio del bucle, el cual controlará el movimiento de nuestro Robot hasta la quinta posición.

5.- Miramos si en la posición en la que nos encontramos actualmente está el bloque **"A"**.

6.- Si la respuesta a la condición puesta en el **Si** es afirmativa pasaremos por esta sexta línea, con lo que cogeremos el bloque.

7.- Finalizamos el **Si**. Seguro que en el momento en el que lo finalizamos tenemos el bloque A en nuestro poder.

8.- Moveremos el Robot una posición a la derecha. Fíjese que nos moveremos tanto a la derecha sin haber encontrado el bloque, como en el caso que ya lo hayamos encontrado.

9.- Pasamos otra vez a comprobar la condición impuesta en el segundo **Mientras**. El programa finalizará siempre en el momento en el que estemos situados en la quinta posición.

En un principio, lo que hemos hecho ha sido mover el Robot a la posición 1 con lo que nos hemos asegurado que ninguna de las posiciones se quedase sin mirar. Puede ser que de esta forma hayamos pasado dos veces por la misma posición pero así nos aseguramos de no dejarnos ninguna.

Cuando ya nos encontramos en la posición 1, vamos pasando posición por posición mirando en cada una de ellas si tenemos el bloque**"A".** Si en la posición que nos encontramos está el bloque **"A"** lo cogemos y seguimos nuestro camino hasta la quinta posición, donde dará por finalizada la ejecución del programa.

15. Las variables

[http://www.mailxmail.com/curso-metodologia-programacion/variables]

Una variable es un "espacio" al cual se le puede asignar diferentes valores según nos convenga. Imaginemos que tenemos una caja a la que le ponemos un nombre para poder hacernos referencia a ella; en la que podemos guardar dentro lo que mejor nos convenga.

En nuestro curso utilizaremos dos tipos de variables: numéricas y de texto, más adelante veremos sus características. Piense que, según el tipo de lenguaje que estemos utilizando para realizar nuestro código, pueden existir muchos tipos de variables diferentes.

¿Como definir una variable?.- Para definir (crear) una variable siempre necesitaremos un nombre con el que nos referiremos a ella durante nuestro "programa", seguido de dos puntos (:) y a continuación el tipo de variable que utilizamos, en nuestro caso: texto o número.

Por ejemplo, podremos definir una variable llamada Escalón en la que guardaremos el número de escalón en el que se encuentra nuestro Robot en un momento determinado. Para definir esta variable deberemos escribir en nuestro código: **Escalón: texto.**

Inicializar una variable.- Antes de empezar a utilizar las variables que hemos definido, es conveniente iniciarlas (darles valores conocidos por nosotros). Para poner un valor a nuestras variables lo haremos de la siguiente manera.

Para una variable donde almacenaremos un número lo único que tendremos que hacer es escribir el nombre de la variable en la que deseemos almacenar el valor, seguido de un signo igual y, a continuación, el número que deseemos almacenar:
Escalón = 10

Para una variable donde almacenaremos caracteres, lo único que tendremos que hacer es escribir el nombre de la variable en la que deseemos almacenar el valor, seguido de un signo igual y a continuación los caracteres que deseemos almacenar, pero en este caso deben ir entre comillas: **NombreRobot = "Arint"** (Aprovechando la ocasión vamos a bautizar a nuestro Robot, a partir de este momento le llamaremos **Arint**)

Observe que una variable numérica y una de texto, sólo se diferencian en que el valor que deseamos guardar, uno no lleva comillas y el otro sí.

Mire detenidamente estos dos ejemplos: **Valor = 10 y Valor = "10"**. Los dos, en un principio, tendrían el mismo valor, pero en realidad no es así. En el primer ejemplo, tendríamos un número con el cual podríamos operar, mientras que en el segundo tendríamos dos caracteres, con los que no podríamos operar.

16. Características de las variables numéricas

[http://www.mailxmail.com/...urso-metodologia-programacion/caracteristicas-variables-numericas]

Con una variable de tipo numérica podremos hacer cualquier tipo de operación, siempre teniendo presente, qué operaciones nos deja hacer nuestro lenguaje de programación.

En este pequeño curso sólo utilizaremos las 4 operaciones básicas (suma, resta, multiplicación y división).

Debemos tener en cuenta que estas operaciones tienen una prioridad. De una cadena de operaciones siempre se realizará, en primer lugar, las divisiones y las multiplicaciones y, en segundo lugar, las sumas y las restas. Si quisiéramos que esta prioridad se viese alterada podríamos utilizar los paréntesis. Las operaciones que aparezcan dentro de ellos se realizarán con una prioridad superior a las que están fuera; sin tener en cuenta si estas son multiplicaciones, divisiones, sumas o restas.

Observe estos dos ejemplos: **1+2*3 = 7** Mientras que **(1+2)*3 = 9**. En el primer ejemplo hemos efectuado primero la multiplicación y después la suma. Mientras que en el segundo, primero se efectúa la suma (por estar entre paréntesis) y después la multiplicación.

Esto es común a la gran parte de lenguajes de programación.

Almacenar resultados en variables.- Nosotros podemos almacenar valores en nuestras variables de la misma manera que las inicializamos. **Variable = Valor.**

Este "almacenamiento" de información la podemos hacer en cualquier lugar del código. Debemos pensar que este **Valor** puede ser sustituido por una operación y almacenar el resultado en la variable.

Veamos un ejemplo: **Variable = 5 + 3** de esta forma cuando nosotros consultemos el contenido de **Variable** podremos observar que es un **8.**

Podemos utilizar una variable como un contador. Un contador no es más que una variable que se va incrementado de forma que podemos contar, por ejemplo, las veces que pasamos por un lugar determinado, las veces que se realiza una función, etc. La estructura de un contador sería la siguiente. Imagine que hemos definido una variable llamada **Contador**. Entonces, el ejemplo quedaría de la siguiente forma: **Contador = Contador + 1**.

Explicaremos cómo funcionaría este contador. A la derecha del igual se produce la operación, que en este caso es **Contador + 1**, de esta forma se coge el valor que tiene **Contador** y se le suma **1**. El resultado de la operación se guarda en la variable que tenemos a la derecha del igual, que este caso es la misma variable **Contador**. Si esta línea la pusiéramos dentro de un bucle, podríamos ver como la **Variable** va aumentando de **1** en **1** hasta que se cumpliera la condición del bucle.

17. Ejemplo 1 - Contador dentro de un bucle

[http://www.mailxmail.com/...curso-metodologia-programacion/ejemplo-1-contador-dentro-bucle]

Veamos un ejemplo en el que utilizaremos una variable, que hará de contador, dentro de un bucle. Volvemos a tener nuestro pequeño Robot Arint, montado en una plataforma móvil de un total de 20 posiciones. Lo que queremos es lo siguiente: el Robot puede partir de cualquiera de las 20 posiciones, queremos que cuando el Robot llegue a la última posición, el programa nos devuelva el número de posiciones por las cuales NO ha pasado.

Antes de mirar la posible solución, intente hacerlo usted mismo.

Vamos a definir una nueva instrucción:

-Mostrar: Arint nos mostrará el valor de la variable o el mensaje que escribamos detrás de esta instrucción.

Es conveniente que revise todas las instrucciones que hemos ido utilizando durante el curso para poder resolver este ejemplo.

Posible solución al ejemplo 1.- Pasemos a la implementación del código y a su correspondiente explicación. Ésta no es la única manera que tenemos para realizar la solución.

```
1.- Contador: Número
2.- Faltan: Número
3.- Contador = 1
4.- Mientras Posición <> 20 hacer
5.-       MoverDer
6.-       Contador = Contador +1
7.- Fin Mientras
8.- Faltan = Posición - Contador
9.- Mostrar Faltan
```

En la próxima lección explicaremos este ejemplo.

18. Explicación del ejemplo anterior

[http://www.mailxmail.com/...curso-metodologia-programacion/explicacion-ejemplo-anterior-2]

Vamos a explicar el ejemplo anterior línea a línea.

1.- Definimos la variable llamada **Contador** como numérica, la cual nos servirá para contabilizar las veces que se mueve **Arint**.

2.- Definimos la variable **Faltan** como numérica, la cual nos servirá para acumular la resta entre el total de posiciones que hay la cantidad de veces que se ha movido **Arint Arint**

3.- Iniciamos la variable **Contador** a **1.** Ya que interpretamos que la posición en la que se encuentra **Arint** se debe contabilizar como posiciones por donde ha pasado.

4.- Iniciamos el bucle, el cual nos servirá para controlar cuando **Arint** llega a la posición 20 (última posición), lugar donde deberá terminar de avanzar.

5.- Mientras no lleguemos a la posición 20, iremos moviendo **Arint** hacia la derecha.

6.- Cada vez que realizamos un movimiento incrementamos en 1 la variable contador. Fíjese, por ejemplo, que la primera vez que entramos en el bucle el valor de la variable **Contador** es **1**. En esta línea le sumamos 1 con lo que el resultado es 2 y queda almacenado (sustituyendo el valor anterior), dentro de la misma variable **Contador**.

7.- Ponemos el punto donde finaliza el bucle para así volver a evaluar la condición.

8.- Calculamos la cantidad de espacios por las que **Arint** no ha pasado. **Restando Posición - Contador** y almacenando el resultado dentro de la variable **Faltan**.

9.- **Arint** por fin nos mostrará por cuántas posiciones en total no ha pasado.

Observe que, en este primer ejemplo, hemos definido dos variables para que la explicación del mismo fuera mucho más claro; ya que, es el primer ejemplo en el que utilizamos variables. El mismo ejercicio que hemos visto se podría haber realizado con una sola variable. Observe cómo se realizaría.

```
Contador: Número
Contador = 1
Mientras Posición <> 20 hacer
    MoverDer
    Contador = Contador +1
Fin Mientras
Contador = 20 - Contador
Mostrar Contador
```

Intente realizar usted mismo la traza de este código.

A la hora de solucionar cualquier problema en el que pensemos que necesitamos utilizar variable, nos tenemos que plantear bien, cuántas utilizaremos y cómo las

utilizaremos. No deberemos ir definiendo variables sin seguir un orden ya que; éstas ocupan espacio y puede hacer que la aplicación vaya más despacio.

19. Estructuras condicionales (II)

[http://www.mailxmail.com/curso-metodologia-programacion/estructuras-condicionales]

Vamos a repasar la estructura condicional que conocemos hasta este momento **"Si",** con un pequeño ejercicio: vamos a hacer que nuestro ya conocido Robot **"Arint"** se vuelva a pasear entre las plataformas móviles. El robot lo situaremos en la primera de 10 posiciones posibles y encima de cada una de ellas habrá o no colocado uno de los bloques con un nombre determinado (A, B ó C), lo que queremos es lo siguiente: el Robot debe pasearse por todas las posiciones hasta llegar a la última, mirando qué bloque está en la posición actual y contabilizando las veces que aparecen los bloques durante todo el recorrido. En la última posición no existirá ningún bloque. Cuando llegue a la última posición nos debe indicar sólo el nombre del bloque que ha aparecido más veces.

Veamos una posible solución de este problema, pero antes de ello y como siempre, es recomendable que intente solucionar por si solo el problema.

```
BloqueA: texto
BloqueB: texto
BloqueC: texto
Mientras Posición <> 10 hacer
    Si BloqueEncima = "A" entonces
        BloqueA = BloqueA + 1
    Fin Si
    Si BloqueEncima = "B" entonces
        BloqueB = BloqueB + 1
    Fin Si
    Si BloqueEncima = "C" entonces
        BloqueC = BloqueC + 1
    Fin Si
    MoverDer
Fin Mientras
Si BloqueA > BloqueB entonces
    Si BloqueA > BloqueC entonces
        Mostrar "Hay más bloques A"
    Si no
        Mostrar "Hay más bloques C"
    Fin si
Si no
    Si BloqueB > BloqueC entonces
        Mostrar "Hay más bloques B"
    Si no
        Mostrar "Hay más bloques C"
    Fin si
Fin si
```

No explico el funcionamiento del código ya que es bastante sencillo de entender y supongo que todos los que habéis seguido el curso hasta este punto lo podréis comprender con facilidad. Es recomendable hacer una traza para poder entender perfectamente el funcionamiento.

Fíjese que, en este ejemplo, para poder saber qué bloque es el que se encuentra encima e incrementar su variable hemos tenido que utilizar 9 líneas en las que hay tres estructuras **Si**, una para cada bloque. Imagine que en lugar de existir tres tipos de bloques hubiéramos podido encontrar 20, ¿qué hubiera pasado con el número de líneas y el número de instrucciones**Si**?

Para facilitar esto, existe una nueva estructura llamada **En caso de**, cuya estructura veremos en la próxima unidad.

20. Nueva estructura condicional

[http://www.mailxmail.com/curso-metodologia-programacion/nueva-estructura-condicional]

Ahora veremos la estructura de la nueva estructura condicional, llamada **En caso de.** Esta estructura, como ya hemos dicho anteriormente, es muy útil en el momento en el que debemos tomar diferentes decisiones de una misma condición.

En caso <CONDICIÓN> hacer
  ...
Fin caso

Vamos a explicar cómo funciona esta nueva estructura condicional.

En el lugar donde aparece la palabra condición pondremos lo que nosotros queremos evaluar: puede ser una variable, una instrucción o cualquier cosa que en nuestro programa tome diferentes valores. Esta condición funciona igual que en la estructura condicional **Si**.

En el lugar de ... pondremos los diferentes valores que pueden darse en la condición. Observe que esta estructura no se suele utilizar cuando la respuesta a la condición es solamente Verdadero o Falso, sino cuando esta condición puede tomar diferentes valores.

Después de estas líneas de valor, escribiremos las instrucciones que queramos que se realicen en cada caso. Piense que estas instrucciones pueden ser sólo una operación o toda una serie de instrucciones anidadas, tan complejas como lo requiera el programa que estamos realizando.

21. Ejemplo de la nueva estructura condicional

[http://www.mailxmail.com/...rso-metodologia-programacion/ejemplo-nueva-estructura-condicional]

Vamos a aplicar esta estructura al ejemplo del principio de la lección:

```
BloqueA: texto
BloqueB: texto
BloqueC: texto
En caso BloqueEncima hacer
     "A"
          BloqueA = BloqueA + 1
     "B"
          BloqueB = BloqueB + 1
     "C"
          BloqueC = BloqueC + 1
Fin Caso
Si BloqueA > BloqueB entonces
    Si BloqueA > BloqueC entonces
        Mostrar "Hay más bloques A"
    Si no
        Mostrar "Hay más bloques C"
    Fin si
Si no
    Si BloqueB > BloqueC entonces
        Mostrar "Hay más bloques B"
    Si no
        Mostrar "Hay más bloques C"
    Fin si
Fin si
```

Observe detenidamente las líneas en las que aparece la nueva estructura y compárela con las líneas correspondientes del primer ejemplo. Observe que aquí, según el valor que toma la condición **BloqueEncima**, se realizan unos u otros pasos. La diferencia de utilización entre una y otra instrucción sería mucho más evidente si los posibles valores que tomara la condición fueran más numerosos.

22. Ejercicio práctico

[http://www.mailxmail.com/curso-metodologia-programacion/ejercicio-practico]

Vamos a realizar nuestro primer programa utilizando un lenguaje de programación real. Para ello, utilizaremos **Visual Basic.**

Veremos cómo nos planteamos el programa en nuestro lenguaje y cómo lo convertimos después a un leguaje real de programación.

Enunciado del ejercicio.- Vamos a hacer un programilla que se semejará un poco a lo que hemos estado haciendo hasta este momento del **Arint** sobre el raíl móvil, para que así, en el momento de hacer nuestro pseudo código, no nos tengamos que inventar ninguna instrucción que no hallamos visto hasta el momento y nos sea mucho más fácil de entender.

Primero explicaremos de qué va el ejemplo y después, como buenos programadores que somos, nos plantearemos cómo enfrentarnos a él realizando el pseudo código.

Antes de continuar explicaremos cuál es el ejemplo real ante el que nos enfrentamos y después haremos un paralelismo hacia nuestro gran amigo **Arint**.

El ejemplo real es el siguiente: tendremos un espacio reservado para poder introducir una frase de un tamaño ilimitado (tampoco deberá ser muy larga) que debe acabar con punto, para que el programa sepa donde finaliza la frase. El programa se iniciará situando un puntero en la primera letra de la frase. Después, dicho puntero se irá moviendo letra a letra hasta llegar al final de la frase, (hasta que el puntero encuentre el punto). Durante este recorrido el programa deberá contar cuantas letras vocales encuentra. Cuando llegue al final nos deberá mostrar la cantidad de vocales de cada tipo que ha encontrado.

Éste es el ejemplo real, ahora veamos el paralelismo con **Arint**: tenemos nuestra plataforma móvil con una cantidad de posiciones ilimitadas. En las diferentes posiciones, siempre desde el principio, pondremos una serie de bloques, cada bloque tendrá una letra asignada. El último de los bloques debe tener un punto para que el Robot sepa donde se encuentra el final. El Robot lo que debe hacer es moverse desde la primera a la última posición contando los bloques que va encontrando por su camino que contengan la letra:**a, e, i, o, u**. Una vez llegado al final deberá mostrarnos cuántas letras de cada tipo ha encontrado por el camino.

Con este ejemplo, es con el que realizaremos los primeros pasos de planteamiento del problema que veremos en la próxima lección.

23. Planteamiento del problema

[http://www.mailxmail.com/curso-metodologia-programacion/planteamiento-problema]

Veamos, a continuación, el planteamiento del problema del que hablábamos en la lección anterior.

1.- Cuando nos encontramos delante de un problema así lo que nos podemos plantear, en un principio, es cómo se finaliza la ejecución del programa y cuáles son las estructuras e instrucciones que se repiten.

En este ejemplo, el proceso que se repite es el movimiento de posición en posición, hasta que se encuentra un bloque con un punto (.). Con lo que nos podemos plantear que todo nuestro programa estará englobado dentro de esta estructura:

```
Mientras BloqueEncima <> "." hacer
   ...
   MoverDer
Fin Mientras
```

Los puntos suspensivos se sustituirán por más instrucciones después de seguir analizando el problema y la instrucción **MoverDer** sabemos que también estará ya que nos tenemos que ir desplazando por las diferentes posiciones.

2.- Después de analizar que es lo que se repite y cómo debe acabar el programa, debemos pensar en que es lo que se debe hacer en cada repetición, dicho de otra manera, cuáles son las instrucciones que se deben hacer en cada una de las posiciones.

Veamos lo que realizará el Robot en cada uno de los pasos:

-Mirar qué bloque tiene encima.

-En caso de ser una de las letras (**a, e, i, o, u**), incrementar en **1** la variable que deberemos crear para cada una de las letras. A estas variables vamos a llamarlas **LetraA, LetraE, LetraI, LetraO, LetraU** que definiremos como siempre al principio del programa. Debemos acordarnos de definir e inicializar las variables que vamos a utilizar en nuestro ejemplo. Esto lo haremos al principio del programa, fuera del bucle ya que sino, las variables siempre se irían reinicializando y no se incrementarían.

-Una vez mirada la letra del bloque que tenemos encima pasaremos a movernos una posición hacia la derecha para volver a empezar los pasos hasta ahora indicados.

24. Cómo implementar lo que hemos aprendido

[http://www.mailxmail.com/...rso-metodologia-programacion/como-implementar-que-hemos-aprendido]

Veamos cómo implementaremos lo hasta aquí explicado.

```
LetraA: Número
LetraE: Número
LetraI: Número
LetraO: Número
LetraU: Número
LetraA = 0
LetraE = 0
LetraI = 0
LetraO = 0
LetraU = 0
Mientras BloqueEncima <> "." hacer
    En caso BloqueEncima hacer
        "a"
            LetraA = LetraA + 1
        "e"
            LetraE = LetraE + 1
        "i"
            LetraI = LetraI + 1
        "o"
            LetraO = LetraO + 1
        "u"
            LetraU = LetraU + 1
    Fin Caso
    MoverDer
Fin Mientras
```

3.- Después de esto, lo único que nos queda es mirar qué es lo que debe hacer el programa justo antes de finalizar.

En nuestro caso, lo único que debe hacer cuando **Arint** encuentre el bloque con el punto es decirnos cuántos bloques con la **a, e, i, o, u** ha encontrado por el camino.

Ahora vamos a añadir las líneas que nos quedan, para completar todo el código. Las líneas que aparecen a continuación deberá ponerlas después del **Fin mientras.**

```
Mostrar LetraA
Mostrar LetraE
Mostrar LetraI
Mostrar LetraO
Mostrar LetraU
```

Después de analizar y plantear un posible código para nuestro pequeño problema vamos a implementarlo en Visual Basic.

25. Implementación en Visual Basic

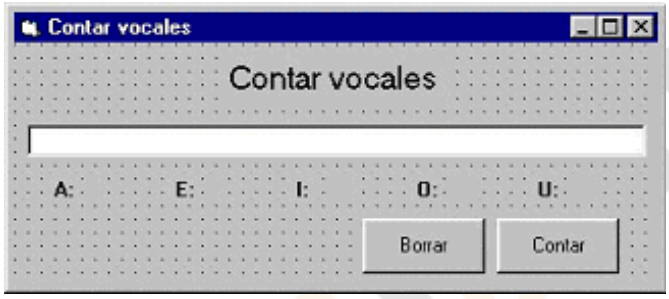
[http://www.mailxmail.com/curso-metodologia-programacion/implementacion-visual-basic]

Una vez que ya tenemos estudiado cómo vamos a plantearnos nuestro problema, veremos cómo queda al traducirlo a Visual Basic.

Visual Basic es un lenguaje de programación en el cual se pueden generar ventanas iguales que la mayoría de programas que se utilizan dentro del entorno Windows.

Una de las primeras cosas que se hacen al realizan un programa en Visual Basic es generar un formulario donde ocurrirá toda la "acción".

Nuestro formulario sería más o menos de este estilo, deberá imaginarse el funcionamiento ya que en este momento no disponemos de los suficientes conocimientos como para poder realizar este ejercicio realmente en Visual Basic.



Vamos a ver los pasos que necesitamos para poder comenzar a trabajar con ese formulario en la siguiente lección

26. Pasos a seguir para la implementación

[http://www.mailxmail.com/curso-metodologia-programacion/pasos-seguir-implementacion

Para poder entender todo lo que vamos a ver, sería mejor que tuviera a mano el código que hicimos en las lecciones anteriores.

Veamos cómo quedaría el código en Visual Basic:

```
LetraA = 0 'Inicializamos todas las variables a 0.
LetraE = 0
LetraI = 0
LetraO = 0
LetraU = 0
Posición = 1 'Situamos el puntero en la primera posición.
Letra = LCase(Frase.Text, Posición, 1) 'Almacenamos en la variable ¿Letra¿ la letra que hay en la posición en la que nos encontramos. Para poder compararla.
Do While Letra <> "." 'Comenzaremos con el bucle hasta que encontremos un punto.
    Select Case Letra 'En caso...
        Case "a" 'En el caso que la letra sea una a...
            LetraA = LetraA + 1 'Sumaremos 1 a la variable LetraA.
        Case "e" 'En el caso que la letra sea una e...
            LetraE = LetraE + 1 'Sumaremos 1 a la variable LetraE.
        Case "i" 'En el caso que la letra sea una i...
            LetraI = LetraI + 1 'Sumaremos 1 a la variable LetraI.
        Case "o" 'En el caso que la letra sea una o...
            LetraO = LetraO + 1 'Sumaremos 1 a la variable LetraO.
        Case "u" 'En el caso que la letra sea una u...
            LetraU = LetraU + 1 'Sumaremos 1 a la variable LetraU.
    End Select 'Fin En caso...
    Posición = Posición + 1 'Instrucción equivalente a MoverDer, ya que incrementamos la posición en 1.
    Letra = LCase(Frase.Text, Posición, 1) 'Almacenamos en la variable Letra la letra que hay en la posición en la que nos encontramos.
Loop 'Finalizamos el bucle mientras.
NumA.Caption = LetraA 'Mostramos la variable LetraA.
NumE.Caption = LetraE 'Mostramos la variable LetraE.
NumI.Caption = LetraI 'Mostramos la variable LetraI.
NumO.Caption = LetraO 'Mostramos la variable LetraO.
NumU.Caption = LetraU 'Mostramos la variable LetraU.
```

Todo lo que aparece después de 'son anotaciones para aclarar las instrucciones' son anotaciones que se ponen en el Visual Basic. El programa no hace caso de ellas.

Intente ver las similitudes con el código que hemos planteado en esta lección y el que hicimos en la lección anterior. Seguramente no entiendan todas las instrucciones que aparecen en este código, pero lo único que pretendemos es que veáis cómo hemos "traducido" el código de un lenguaje que nos sirve para plantearnos el problema a

otro real.

27. Bucles (II)

[http://www.mailxmail.com/curso-metodologia-programacion/bucles-2]

Como hemos visto hasta este momento, los bucles son una de las instrucciones más utilizadas dentro del mundo de la programación ya que, muchos de los problemas con los que nos encontramos y queremos que resuelva un ordenador suelen ser rutinas repetitivas.

Hasta este momento, hemos visto la instrucción **Mientras... Fin Mientras**, pero esta nos puede traer un pequeño problema que deberíamos solucionar con líneas adicionales fuera del bucle, veamos esto con un ejemplo.

Tenemos a **Arint**, subido en una plataforma con 10 posiciones, en algunas de estas posiciones tendremos puestos diferentes bloques, incluidas la primera y última posición. **Arint**, para facilitar la cosa, siempre partirá de la primera posición. El Robot deberá recorrer cada una de las posiciones mirando si hay un bloque llamado **A** en ella. Al llegar al final y después de mirar si hay bloque en la posición en la que se encuentra nos deberá decir cuántos bloques ha encontrado por el camino. Recordemos que el Robot nunca podrá salir de las 10 posiciones indicadas.

Utilizando las instrucciones que hemos visto hasta el momento tendríamos que realizar un código parecido a éste:

```
NumBloques: Número
NumBloques = 0
Mientras Posición <> 10 hacer
      Si BloqueEncima = "A" entonces
          NumBloque = NumBloque + 1
      Fin Si
       MoverDer
Fin mientras
Si BloqueEncima = "A" entonces
     NumBloque = NumBloque + 1
Fin si
Mostrar NumBloque
```

Mire detenidamente el código anterior. Fíjese que las instrucciones que tenemos dentro del **Mientras**, hacen que el brazo se mueva de la posición **1** a la **10**, pero sólo miran los bloques que hay en las posiciones de la **1** a la **9**, ya que cuando llega a la posición **10** la condición ya es **VERDADERA**, con lo que no entra. Por eso, hemos hecho que al final del bucle mirase nuevamente si el bloque que hay en la posición **10** tiene la letra **A**.

28. Nueva estructura de bucles

[http://www.mailxmail.com/curso-metodologia-programacion/nueva-estructura-bucles]

Ahora vamos a ver el mismo ejemplo de esta lección, pero utilizando una nueva estructura repetitiva (bucle). Primero veamos la estructura y después cómo la podemos aplicar.

Repetir
...

<INSTRUCCIONES>

Hasta que <CONDICIÓN>

Esta es otra estructura de bucle pero tiene un pequeño matiz, que la hace diferente a la ya estudiada anteriormente.

Observe que esta estructura primero ejecutaría las instrucciones y después miraría si se cumple o no la condición, con lo que tenemos que tener mucho cuidado ya que las instrucciones, como mínimo, se ejecutarán una vez (las veces restantes ya dependerán de la condición). La condición se evalúa después de realizar las instrucciones y no antes, como pasaba en el caso del Mientras.

Ejemplo.- vamos a aplicar esta estructura al ejemplo del principio de la lección:

```
NumBloque: Número
NumBloque = 0
Repetir
     Si BloqueEncima = "A" entonces
          NumBloque = NumBloque + 1
     Fin Si
     MoverDer
Hasta que Posición = 10
Mostrar NumBloque
```

Observe detenidamente las líneas en las que aparece la nueva estructura.

Fíjese que en un principio, después de crear la variable e inicializarla a 0, lo único que ve el programa es que entra en una estructura repetitiva pero no lleva a cabo ninguna verificación de ningún tipo, con lo que ejecuta las dos instrucciones que están dentro de ella. Cuando llega a la línea Hasta que... es cuando el programa mira a su alrededor para ver si se encuentra en la posición 10. Observe también como en este caso es necesario preguntar por si la posición es **IGUAL (=)** y no **DIFERENTE (<>)** como en el caso de la estructura Mientras. Por lo que, cuándo Arint se encuentra en la posición 10 terminará de ejecutarse el bucle, pero con la diferencia, de que aquí ya habrá mirado si existe algún bloque A en la última posición y no necesitaremos volver a preguntar si en esa posición existe un bloque llamado A.

Ahora que hemos analizado y comparado este ejemplo, seguro que tiene muy claras las dos estructuras repetitivas. Vuelvo a insistir en que analice bien cuál de las dos estructuras deberemos utilizar en cada caso y si creemos que debemos utilizar un bucle.

29. Operadores lógicos

[http://www.mailxmail.com/curso-metodologia-programacion/operadores-logicos]

Hasta este momento hemos visto que tanto en una condición (**Sí... o En caso de...**) como en un bucle **(Mientras...)** utilizamos una sola condición para llevarla a cabo. Pero puede ser que nos interese valorar más de una condición. Me explico: puede ser que en un determinado momento nos interese mirar si el bloque que tenemos encima se llama **A** y nos encontramos en la posición **5**, con lo que podríamos tomar una decisión determinada. Otra cosa que podríamos mirar es si el bloque se llama **A** o nos encontramos en la posición **5**. Fíjese que estos dos ejemplos tienen una pequeña diferencia, uno esta planteado utilizando una **o** y otro una **y**, con lo que el resultado es diferente. A esta **o** y a esta **y** se les llama operadores lógicos.

Analizando el operador lógico Y.- Para analizar el funcionamiento de este operador lógico vamos a plantear un pequeño caso para ver qué es lo que ocurriría en diferentes momentos.

Tenemos a **Arint** moviéndose libremente por nuestra pasarela con 10 posiciones y queremos que mire si se encuentra en la posición **5** y si el bloque que tiene encima se llama **A**. Si es así, deberá cogerlo y llevarlo a la primera posición.

En un principio, podríamos pensar en utilizar un **Si** dentro de otro, pero como ya hemos dicho muchas veces, cuantas menos líneas tenga el código, mejor. Veamos igualmente cómo sería el código utilizando un **Si** anidado con otro. En un principio, en el primero de los **"Si's"** podemos preguntar por cualquiera de las condiciones que se deben cumplir y dentro por la que queda. Dentro de los dos **"Si's"** pondremos las líneas que se debe ejecutar en caso de que las dos condiciones sean **Verdaderas**. (Sólo escribiremos las líneas en las que se utiliza la instrucción **Si**).

```
Si Posición = 5 entonces
    Si BloqueEncima = "A" entonces
        Instrucciones a realizar...
    Fin Si
Fin Si
```

Éste sería el código con un **Si** dentro de otro. ¿Qué es lo que haría el programa cuando llegara a estas líneas? Primero miraría la primera condición si se cumple. Si es verdadera, pasaría a la siguiente línea, miraría si esta condición se cumple. En caso afirmativo, realizaría las instrucciones que hubiera dentro. Si cualquiera de las dos condiciones que tenemos en este ejemplo no se cumplieran el programa no realizaría las instrucciones internas.

30. Cómo implementar funciones dentro de un mismo 'Si'

[http://www.mailxmail.com/...todologia-programacion/como-implementar-funciones-dentro-mismo-s

Ahora vamos a ver cómo podemos implementar las mismas funciones pero todo dentro de un mismo **Si**. En un principio, la estructura del **Si** es exactamente igual, lo único que debemos pensar es que tendremos que separar las dos condiciones mediante un nexo. En este caso, como queremos que se cumplan las dos condiciones utilizaremos un nexo **Y**. Veamos cómo quedaría el ejemplo y después pasaremos a comentarlo viendo su funcionamiento en diferente casos.

```
Si Posición = 5 Y BloqueEncima = "A" entonces
    Instrucciones a realizar...
Fin Si
```

Observe cómo el número de líneas se ha reducido. Ahora vamos a plantear las diferentes opciones con las que nos podemos encontrar y si realizaría o no las instrucciones que se encuentran dentro del **Si**. Recuerde que las condiciones sólo pueden ser **Verdaderas** o **Falsas**.

Posición = 5	BloqueEncima = "A"	¿Se realizarían las instrucciones?
Verdadero	Verdadero	Sí (Verdadero)
Verdadero	Falso	No (Falso)
Falso	Verdadero	No (Falso)
Falso	Falso	No (Falso)

Observe que el único caso en el que se realizarían las instrucciones sería en el que se cumplen las dos condiciones planteadas en el **Si.** Por el contrario, si una de las dos instrucciones no se cumple las instrucciones no se ejecutarían. Fíjese bien en el cuadro e inténtelo entender.

31. Analizando el operador lógico O

[http://www.mailxmail.com/curso-metodologia-programacion/analizando-operador-logico]

Para analizar el funcionamiento, de este otro operador lógico vamos a plantear otro pequeño caso para ver que es lo que ocurriría en diferentes momentos.

Tenemos a **Arint**, otra vez, moviéndose libremente por nuestra pasarela con 10 posiciones y queremos que mire si el bloque que tiene encima se llama **A** o si se llama **B**. Si es así, deberá coger el bloque y llevarlo a la primera posición.

Para poder realizar este pequeño ejemplo sin utilizar los operadores lógicos deberíamos utilizar dos instrucciones **Si** de la siguiente manera:

```
Si BloqueEncima = "A" entonces
   Instrucciones a realizar...
Fin Si
Si BloqueEncima = "B" entonces
   Instrucciones a realizar...
Fin Si
```

¿Que es lo que haría **Arint** cuando llegara a estas líneas? Primero miraría el primer **Si**, si se cumpliera la condición, realizaría las instrucciones que tenemos en el interior, sino pasaría al siguiente **Si** y miraría la siguiente condición. Aquí, si esta condición se cumple se realizan las instrucciones que tenemos en el interior. De tal manera, si una de las dos condiciones se cumple se realizarían las instrucciones. Piense que no podemos utilizar un **Sino** dentro del primer **Si** ya que podría ser que nos encontrásemos con un bloque llamado **C,** con lo que también se realizarían las instrucciones, cosa que no nos interesa. Por eso, ponemos dos **Si** para comprobar que el bloque de encima se llama **A** o se llama **B.**

Ahora vamos a ver cómo podemos implementar las mismas funciones pero todo dentro de un mismo **Si**. En un principio, la estructura del **Si** es exactamente igual, lo único que debemos pensar es que tendremos que separar las dos condiciones mediante un nexo. En este caso, como queremos que se cumpla una de las dos condiciones utilizaremos un nexo **O**. Veamos cómo quedaría el ejemplo y después pasaremos a comentarlo viendo su funcionamiento en diferentes casos.

```
Si BloqueEncima = "A" O BloqueEncima = "B" entonces
    Instrucciones a realizar...
Fin Si
```

Observe como el número de líneas se ha reducido y más pensando que tendríamos que repetir las instrucciones internas dos veces en los dos **Si's.** Ahora vamos a plantear las diferentes opciones con las que nos podemos encontrar y si realizaría o no las instrucciones que se encuentran dentro del **Si**. Recuerde que las condiciones sólo pueden ser **Verdaderas** o **Falsas**.

BloqueEncima = "A"	BloqueEncima = "B"	¿Se realizarían las instrucciones?
Verdadero	Verdadero	Sí (Verdadero)
Verdadero	Falso	Sí (Verdadero)
Falso	Verdadero	Sí (Verdadero)
Falso	Falso	No (Falso)

Observe que el único caso en el que no se realizarían las instrucciones sería en el que no se cumple alguna de las dos condiciones planteadas en el **Si**. De forma que si una de las dos condiciones es verdadera las instrucciones se realizan. Fíjese bien en el cuadro e inténtelo entender.

32. Explicación matemática

[http://www.mailxmail.com/curso-metodologia-programacion/explicacion-matematica]

Vamos a ver cómo podemos entender estos conceptos de operadores lógicos de forma matemática. Esto es interesante en el momento en el que nos podemos plantear una mezcla de **Y** y de **O** , dentro de un mismo **Si** u otra instrucción. Más adelante, verá a lo que me refiero exactamente.

Antes de continuar explicaremos que un ordenador realiza los cálculos en sistema binario y no en sistema decimal como nosotros, sólo utiliza dos dígitos: el **1** y el **0**. Con estos dos dígitos, representa cualquier número. Vamos a realizar una comparación entre el sistema decimal y el binario contando del **1** al **10**.

Decimal	Binario
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

No vamos a comentar cómo ¿funciona¿ el código binario al completo, ya que sería un poco largo y tedioso. Puede ser que más adelante lo hagamos ya que es muy interesante. Lo único en lo que queremos hacer hincapié es en un par de operaciones, la suma y la multiplicación; más adelante veremos para qué nos pueden servir. En un principio, lo único que nos interesaría sumar y multiplicar serían **1** y **0**, por lo que el tema se convierte en un poco más sencillo.

Veamos primero la suma, cómo sería en decimal y cómo es en binario.

Decimal	Binario
1 + 1 = 2	1 + 1 = 10
1 + 0 = 1	1 + 0 = 1
0 + 1 = 1	0 + 1 = 1
0 + 0 = 0	0 + 0 = 0

Observe que en la parte de la operación en Binario lo único que se ha visto afectado es el número **2** que, como ya hemos visto en la tabla anterior, en Binario es un **1 0**. Una vez hecha esta tabla, compárela con la tabla que realizamos del operador lógico **O** **O** y verá que si sustituimos los **1** por **Verdadero** y los **0** y los **1 0** por **Falso**, la tabla es exactamente igual.

33. La multiplicación en decimal y en binario

[http://www.mailxmail.com/...curso-metodologia-programacion/multiplicacion-decimal-binario]

Veamos ahora la multiplicación cómo sería en decimal y cómo es en binario.

Decimal	Binario
1 x 1 = 1	1 x 1 = 1
1 x 0 = 0	1 x 0 = 0
0 x 1 = 0	0 x 1 = 0
0 x 0 = 0	0 x 0 = 0

Observe que en la parte de la operación en Binario no se ha visto afectado de ninguna manera ya que no se han utilizado dígitos que no sean el **1** y el **0**. Una vez hecha esta tabla, compárela con la tabla que realizamos del operador lógico **Y** y vera que si sustituimos los **1** por **Verdadero** y los **0** por **Falso**, la tabla es exactamente igual.

En definitiva, lo que queremos dar a entender con esto es que en el momento en el que nos veamos delante de la necesidad de utilizar una combinación de **Y** y **O** dentro de una misma condición, podemos utilizar la multiplicación y la suma en binario para poder ver cuál es el resultado que nos daría.

34. ¿Qué es una tabla?

[http://www.mailxmail.com/curso-metodologia-programacion/que-es-tabla]

En un principio, no vamos a poner una definición, ya que lo que interesa es que se entienda el concepto de forma lógica. Para esto, sólo hace falta hacer una pequeña comparación con los ejemplos que hemos utilizado en todo este curso.

Podemos decir que una tabla no es otra cosa que toda la estructura por la cual se ha ido moviendo nuestro incansable **Arint**. O sea, una serie de posiciones en las que se podían "poner" cosas de un mismo tipo, que en nuestros ejemplos eran **Bloques**. También podemos decir que esos bloques, a parte de por su nombre, se diferenciaban entre sí por la posición que ocupaban.

Una tabla se define con un nombre y una serie de posiciones y un tipo de dato que irá dentro de esta estructura. Haciendo la similitud con nuestro ejemplo esto quedaría así: nuestro **Arint** se ha movido por una estructura que no tenía nombre pero le podemos asignar uno llamándola **Cinta**. Después, nosotros siempre definíamos cuantas posiciones tenía esta estructura, pues esto es el tamaño de la tabla. Se puede decir que esta estructura tiene una posición mínima y otra máxima, a esto le llamaremos **Rango de la tabla**. En los ejemplos, no definíamos qué tipo de datos se podía encontrar en la estructura pero normalmente trabajábamos con **Bloques**, pues bien, esto sería el tipo de dato que tenemos dentro de la tabla.

Cómo creamos una tabla.- Antes de nada, veamos cómo definiremos una tabla. Recuerde los elementos que necesitamos para ello: el nombre, el tamaño (número de posiciones), y el tipo que tendrán los datos que se almacenarán en el interior de la tabla.

Imaginemos que queremos definir una tabla llamada: **Números** con 10 posiciones y que sea de tipo Numérico. En este caso, la definición sería de la siguiente forma **N Números[10] : Número.**

Cada elemento de esta tabla se diferencia de otro elemento de la misma por el lugar que ocupa, o sea, por su posición en la tabla, al igual que hacíamos con **Arint** cuando le preguntábamos la posición en la que se encontraba.

De esta forma, el primer elemento sería **Números[1]** y el último **Números[10].** En muchos lenguajes de programación el primer elemento de una tabla ocupa el lugar **0 0**, pero nosotros, para facilitar el trabajo, empezaremos a contar desde el **1.**

35. Operando con tablas

[http://www.mailxmail.com/curso-metodologia-programacion/operando-tablas]

Cada elemento de una tabla se comporta de forma independiente con respecto a los demás, con esto quiero decir que cada elemento podemos pensar que se trata, en un principio, de una variable diferente. De forma que, si nosotros queremos hacer una operación entre el primer y el tercer elemento de nuestra tabla podríamos hacerlo de la siguiente forma: podríamos crear una variable llamada **Operación** para almacenar el resultado de la operación, siendo la instrucción de la siguiente forma: **Operación = Números[1] + Números[3].**

Primer Ejemplo.- Ahora vamos con el ejemplo antes mencionado: Imagínese que tenemos 200 números al azar almacenados en variables y lo que deseamos es obtener el doble del valor que tienen actualmente.

Piense que si en un principio tuviéramos que definir 200 variables e ir multiplicando cada una de ellas por el doble sería un poco largo, y cuantas más variables tuviéramos, más largo sería el código. Pues bien, si utilizamos una tabla esto es algo poco más fácil.

Enumeraremos las líneas para después poder hacer la explicación paso a paso. El código quedaría de la siguiente manera:

```
1.- Valor[10]: Número
2.- Indice: Número
3.- Indice = 1
4.- Mientras Indice < 11 hacer
5.-       Valor[Indice] = Random(100)
6.-        Indice = Indice + 1
7.- Fin Mientras
8.- Indice = 1
9.- Mientras Indice < 11 entonces
10.-       Valor[Indice] = Valor[Indice] * 2
11.-        Indice = Indice + 1
12.- Fin Mientras
```

En la próxima lección le explicaremos cada una de las líneas del ejemplo

36. Explicación al ejemplo anterior

[http://www.mailxmail.com/...curso-metodologia-programacion/explicacion-ejemplo-anterior-3]

Veamos a continuación, la explicación al ejercicio anterior.

1.- Definimos una tabla llamada **Valor** con 10 posiciones de tipo número.

2.- Definimos una variable llamada **Índice** que nos servirá para movernos por el interior de la tabla.

3.- Inicializamos la variable a **1** para colocarnos en la primera posición de la tabla recién creada.

4.- Utilizaremos un bucle para llenar la tabla con unos valores iniciales. El bucle se repetirá mientras el índice sea menor de **11**.

5.- Gracias al bucle iremos pasando por cada una de las posiciones de la tabla introduciendo un número, al azar del uno al 100, utilizando la nueva orden **Random**.

6.- Incrementamos en **1** el **Índice** para poder pasar a la siguiente posición.

7.- Terminamos el bucle para el rellenado de datos.

8.- Volvemos a inicializar la variable a 1 para colocarnos al principio de la tabla.

9.- Volvemos a utilizar un bucle para recorrer toda la tabla. En esta ocasión lo haremos para doblar el valor de cada una de las posiciones.

10.- En esta línea multiplicamos por dos el valor del lugar en el que nos encontramos.

11.- Incrementamos en 1 el **Índice** para poder pasar a la siguiente posición.

12.- Terminamos el bucle.

Intente comprender paso a paso este ejemplo. Plantéese casos parecidos, hasta que comprenda bien cómo nos movemos por las diferentes posiciones de una tabla.

Nota importante.- debe tener mucho cuidado que en los programas en los que utilice tablas no acceda a una posición que esté fuera del rango de la tabla. Si sucediera esto nos daría un error y el programa terminaría su ejecución de forma incorrecta.

37. Segundo Ejemplo

[http://www.mailxmail.com/curso-metodologia-programacion/segundo-ejemplo]

Vamos a ver otro pequeño ejemplo para tratar con tablas. En esta ocasión, tenemos una tabla con 25 elementos que suponemos que ya está llena. Lo que queremos es dar la "vuela" a la tabla, digámoslo de otra manera, el elemento que está en la primera posición ha de ocupar la última y el elemento de la última deberá pasar a la primera. Ej.: La tabla: 1, 2, 3, 4, 5 después tendrá este aspecto 5, 4, 3, 2, 1. Hemos añadido números a las líneas para facilitar la explicación.

```
1.- Tabla[25]: Número
2.- IndicePrimero: Número
3.- IndiceUltimo: Número
4.- Elemento: Número
5.- IndicePrimero = 1
6.- IndiceUltimo = 25
7.- Repetir
8.-     Elemento = Tabla[IndicePrimero]
9.-     Tabla[IndicePrimero] = Tabla[IndiceUltimo]
10.-    Tabla[IndiceUltimo] = Elemento
11.-    IndicePrimero = IndicePrimero + 1
12.-    IndiceUltimo = IndiceUltimo - 1
13.-Hasta que IndicePrimero = IndiceUltimo
```

1.- Definimos una tabla llamada **Tabla** con 25 posiciones de tipo Número.

2.- Definimos una variable llamada **IndicePrimero** que nos servirá para movernos por la tabla desde la primera posición en adelante.

3.- Definimos una variable llamada **IndiceUltimo** que nos servirá para movernos por la tabla desde la última posición hacia atrás.

4.- Definimos una variable llamada **Elemento** para utilizarla de puente en el momento de mover el contenido de una posición a otra. Esta variable es necesaria ya que si hacemos el cambio desde, por ejemplo, la primera a la última posición perderíamos el valor que teníamos en la primera posición.

5 y 6.- Inicializamos las variables **IndicePrimero** a 1 y **IndiceUltimo** a 25 ya que queremos partir, respectivamente, desde la primera y la última posición.

7.- Entramos dentro del bucle.

8.- Hacemos que la variable **Elemento** tome el valor del contenido de la tabla que nos indique **IndicePrimero**.

9.- Hacemos que la posición que nos indica **IndicePrimero** tome como valor el valor que tiene la posición que nos indica **IndiceUltimo**.

10.- Ahora la posición que nos indica **IndiceUltimo** toma como valor el contenido de la variable **Elemento**.

11.- Aumentamos **IndicePrimero** en 1 para adelantar en la tabla.

12.- Disminuimos **IndiceUltimo** en 1 para retroceder en la tabla.

13.- Repetiremos el bucle hasta que los dos índices se encuentren. Éste será el punto en el que no se tendrán que mover más elementos ya que, nos encontramos justo en la mitad de la tabla.

Hay que tener en cuenta que esta solución es válida en el caso que la cantidad de posiciones en la tabla sean impares. Para una cantidad par tendríamos que buscar otra solución. No estaría plantearse esto como un nuevo ejercicio.

Cuando salimos del bucle la ordenación de la tabla ha concluido.

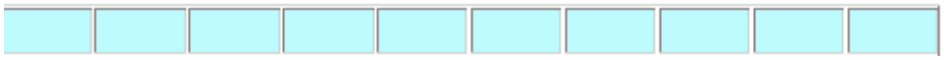
Mírese el código tantas veces como sea necesario, si no entiende algún punto no dude en preguntarlo.

38. Las matrices

[http://www.mailxmail.com/curso-metodologia-programacion/matrices]

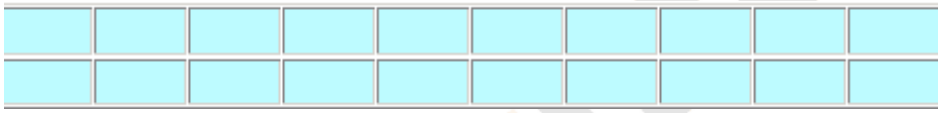
En un principio, podemos decir que una matriz no es otra cosa que una tabla, como las que hemos visto en la lección anterior, pero en dos dimensiones.

Si nosotros representamos una tabla de 10 posiciones lo podríamos hacer así:



En definitiva, en la tabla anterior tenemos 1 fila (horizontal) y 10 columnas (vertical), esto hace un total de 10 posiciones. Con lo que para referirnos a una posición sólo hace falta indicar la columna a la que nos estamos refiriendo. **Valor[3]** Hacemos referencia a la tercera posición de la tabla llamada **Valor**.

Ahora bien, en según que caso, nos puede interesar tener lo que llamaremos matriz, una "tabla" pero con varias filas. Podríamos representar, por ejemplo, una matriz de 2 filas con 10 columnas de la siguiente manera:



En esta matriz, tendríamos 20 posiciones 2 x 10. Para referirnos a ellas tendremos que buscar una manera diferente a como lo hemos hecho con una tabla. En realidad, no es que la forma de llamar a cada una de las posiciones sea muy diferente, pero se añade un elemento más. Recuerde que cuando utilizábamos una tabla lo que hacíamos era indicar la columna que ocupaba el elemento **Datos[4],** pues bien, ahora nosotros deberemos indicar la fila y la columna que ocupa. Esto lo haríamos de la siguiente manera: para referirnos a una posición siempre, al igual que en el caso de una tabla, pondremos el nombre de la tabla entre corchetes [ ], primero la fila y después la columna separados por una coma de la siguiente forma: **Datos[1, 4]**. De esta manera, observe que los elementos que ocupan la cuarta columna se deben diferenciar entre ellos por la fila en la que se encuentran.

Observa este pequeño ejemplo en el que rellenaremos algunos campos de una matriz de 3 filas por 4 columnas, llamada **Valor** en la que introducimos los siguientes valores **Valor[1,3] = "Excel", Valor[2,4] = "Word", Valor[3,2] = "VB".** La tabla quedaría de la siguiente manera, recuerde que siempre miraremos primero la fila (horizontal) y después la columna (vertical):

Valor	1	2	3	4
1			Excel	
2				Word
3		VB		

Le recomiendo que utilice ejemplos como este para que se acostumbres a utilizar

tablas y a colocar los elementos en su lugar, piense que en programación si colocamos los elementos de una matriz de forma incorrecta o nos referimos a una posición fuera de la matriz nos puede dar error y "fastidiar" la ejecución del programa.

39. Cómo creamos una matriz

[http://www.mailxmail.com/curso-metodologia-programacion/como-creamos-matriz]

La forma de crear una matriz es exactamente igual que en el caso de una tabla, con el único cambio que en este caso debemos indicar también cuántas filas forman parte de esta. Recuerde los elementos que necesitamos para ello: el nombre, el tamaño (número de posiciones, tanto filas como columnas), y el tipo que tendrán los datos que se almacenarán en el interior de la matriz.

Imaginemos que queremos definir una tabla llamada **Valores** con 10 columnas y 5 filas que sea de tipo Numérico, esto hace un total de 5 filas por 10 columnas un total de: 50 posiciones. Pues bien, la definición sería de la siguiente forma **Valores[5,** : **Valores[5, 10]: Número**. Observa que en un principio todas las filas que definamos dentro de una misma matriz tendrán el mismo tipo de dato.

Primer Ejemplo.- En muchos lenguajes de programación las tablas, tanto en filas como en columnas empiezan a contar desde la posición 0, esto es importante porque si definimos una tabla de 5 columnas nos debemos referir a ellas desde la posición **0** a la **4**. En nuestro caso, para facilitar el entendimiento de los ejemplo vamos a utilizar matrices cuya primera posición es la número **1.**

Vamos con un ejemplo en el que utilizaremos una matriz: queremos un pequeño programa que nos ordene unos números, que obtenemos de forma aleatoria, en pares e impares. Vamos a definir una tabla con dos filas, de esta manera pondremos en la primera fila (fila 1) los número impares y en la segunda fila (fila 2) los número pares.

Para saber si un número es par o impar lo único que deberemos hacer es dividir la cantidad entre 2 y mirar el resto. Si este resto es**0** querrá decir que el número es par y si el resto es diferente a **0** el número será impar. Para mirar el resto utilizaremos una nueva instrucción: **Resto( )**, poniendo dentro del paréntesis la división a realizar.

```
1.- Tabla[2,10]: Número
2.- IndicePar: Número
3.- IndiceImpar: Número
4.- Cantidad: Número
5.- IndicePar = 1
6.- IndiceImpar = 1
7.- Mientras (IndicePar + IndiceImpar) < 21 hacer
8.-      Cantidad = Random(100)
9.-      Si Resto(Cantidad/2) = 0 y IndicePar < 11 entonces
10.-          Tabla[2,IndicePar] = Cantidad
11.-          IndicePar = IndicePar + 1
12.-     Fin Si
13.-     Si Resto(Cantidad/2) <> 0 y IndiceImpar < 11 entonces
14.-          Tabla[1,IndiceImpar] = Cantidad
15.-          IndiceImpar = IndiceImpar + 1
16.-     Fin Si
```

17.- Fin Mientras

En la próxima lección veremos la explicación a este ejemplo.

40. Explicación al ejemplo anterior

[http://www.mailxmail.com/...curso-metodologia-programacion/explicacion-ejemplo-anterior-4]

Veamos la explicación correspondiente a la lección de cómo crear una matriz.

1.- Definimos una tabla llamada **Tabla** con 2 filas y 10 posiciones de tipo número.

2 y 3.- Definimos una variable llamada **IndicePar** que nos servirá para movernos por la fila de los pares y otra variable llamada **IndiceImpar** para movernos por la fila de los impares.

4.- Definimos una variable llamada **Cantidad** para almacenar el valor del número buscado de forma aleatoria.

5 y 6.- Inicializamos las variables **IndicePar** e **IndiceImpar** a 1 para empezar a rellenar la matriz desde la primera columna.

7.- Iniciamos un bucle que se repetirá hasta que se llene completamente la matriz. Como nuestra matriz tiene 2 filas y 10 posiciones cada una de ellas en total tenemos 20 posiciones.

8.- Almacenamos en la variable **Cantidad** un número aleatorio del 1 al 100.

9.- Miramos si la **Cantidad** es par mediante la formula **Resto(Cantidad/2)** y si el índice que utilizamos para movernos por la fila de los números par no esté fuera de rango, no sea más grande de 10. Recordemos que si intentamos acceder a una posición fuera de la matriz nos daría un error de ejecución.

10.- Si la **Cantidad** es par y el **IndicePar** está dentro de los límites almacenamos **Cantidad** en la matriz **Tabla** en la segunda fila, reservada para los números pares, y en la posición que indique el **IndicePar**.

11.- Incrementamos en 1 el valor de la variable **IndicePar**, para avanzar una posición en la fila de los pares.

12.- Línea que nos indica el fin del primer **Si**.

13.- Miramos si la **Cantidad** es impar mediante la formula **Resto(Cantidad/2)** y si el índice que utilizamos para movernos por la fila de los números impares no esté fuera de rango, no sea más grande de 10.

14.- Si la **Cantidad** es impar y el **IndiceImpar** está dentro de los límites almacenamos **Cantidad** en la matriz **Tabla** en la primera fila, reservada para los números impares, y en la posición que indique el **IndiceImpar**.

15.- Incrementamos en 1 el valor de la variable **IndiceImpar**, para avanzar una posición en la fila de los impares.

16.- Línea que nos indica el fin del segundo **Si**.

17.- Fin del bucle.

Aquí llegamos al final de este curso de introducción a la programación. Todavía faltarían algunas cosas para ver, pero con lo visto hasta el momento nos podemos

hacer una buena idea de qué es la programación y cómo plantearnos muchos problemas que nos puedan ir surgiendo.

Recuerde la importancia de realizar bien las trazas ya que nos pueden facilitar mucho el trabajo y nos pueden ayudar a encontrar errores que podamos cometer y para plantearnos nuevas soluciones.

Le recomiendo hacer muchos ejercicios para tener más soltura en el momento de resolver los programas que nos puedan ir surgiendo.

Cursos similares

Cursos	Valoración	Alumnos	Vídeo
SQL SQL (Structured Query Language) es un lenguaje de programación para acceder y manipular bases de datos. SQL surgió de un proyecto de IBM en el que investigaba e... [10/05/04]		43.949	
Manual de programación El objetivo de este curso consiste en ofrecer conocimientos básicos de programación. No pretender enseñar cómo programar en un lenguaje específico, ni utilizar ninguna te... [31/01/08]		2.801	
Primeros pasos con XML y XSL XML es el acrónimo del inglés eXtensible Markup Language cuyo objetivo principal es conseguir una página web más semántica. Inicialmente nace como sucesor del HTML, separ... [10/09/04]		7.410	
C# Curso de Programación Curso Básico de programación en C#, este curso básico abarcará desde las bases del lenguaje hasta nuestros primeros pasos con aplicaciones web, acceso a bases de datos de... [14/07/05]		10.393	
Funciones en C (primera parte) Curso de programacion informatica sobre Funciones en C en el orden de los fundamentos de la programación. Comprende el desarrollo de un software utilizando funciones y a ... [17/10/08]		5.595	