Tema I. Introducción al Análisis de Algoritmos.

Análisis de Algoritmos.

Analizar un algoritmo es estudiar su comportamiento con el fin de determinar cual de entre varios algoritmos que resuelven un problema es el más eficiente. La solución a un problema no está limitada a un solo algoritmo ya que cada programador puede presentar soluciones diferentes. El poder detectar ineficiencia es uno de los puntos de este curso y para ello hemos seleccionado estratégicamente ejercicios que nos ayudaran poco a poco a encontrar o determinar la solución mas optima según el caso.

Concepto de eficiencia.

Cuando se habla de eficiencia de un algoritmo en cierto sentido el mayor énfasis esta puesto en el tiempo de ejecución del algoritmo. En un sentido amplio, eficiencia se refiere a la forma de administración de todos los recursos disponibles en el sistema, de los cuales el tiempo de procesamiento es uno de ellos.

Análisis de un algoritmo según el tiempo de ejecución.

Cuando se habla de tiempo de ejecución se consideran más eficientes aquellos algoritmos que cumplan con la solución del problema en el menor tiempo posible.

Tiempo de ejecución de un algoritmo.

Hay dos formas de estimar el tiempo de ejecución de un algoritmo y estas son el análisis teórico y el análisis empírico. En el teórico se busca obtener una medida del trabajo realizado por el algoritmo a fin de obtener una estimación teórica de su tiempo de ejecución. Básicamente se calculan el número de comparaciones y de asignaciones que requiere el algoritmo. En el análisis empírico se hace el enfoque al tiempo de respuesta. La aplicación de los mismo datos a distintas soluciones del mimo problema presupone la obtención de una herramienta de comparación entre ellos. El análisis empírico tiene la ventaja de ser muy fácil de implementar, pero no tiene en cuenta algunos factores como:

.**La velocidad de la maquina**, esto es, la ejecución del mismo algoritmo en computadoras distintas produce diferentes resultados. De esta forma no es posible tener una medida de referencia.

.**Los datos con los que se ejecuta el algoritmo**, ya que los datos empleados pueden ser favorables a uno de los algoritmos, pero pueden no representar el caso general, con lo cual las conclusiones de la evaluación pueden ser erróneas.

Análisis de un algoritmo según la administración o uso de la memoria.

En este enfoque serán eficientes aquellos algoritmos que utilicen las estructuras de datos de manera de minimizar la memoria ocupada. En este punto encajan muy bien los problemas donde el énfasis esta puesto en el volumen de la información a manejar en la memoria tales como: manejo de bases de datos, procesamiento de imágenes en memoria, etc.

De estos dos conceptos de eficiencias a lo largo de este curso nos enfocaremos en el primero.

Ejercicios: Estos ejercicios primero aprenderemos a resolverlos manualmente y luego aprenderemos a diagramarlos, pseudo codificarlos y posteriormente a programarlos realizando en cada uno de ellos un análisis para entender como se desarrolla su flujo. Se hará mucho énfasis en la técnica "Divide y vencerás" para poder llegar satisfactoriamente a la solución deseada y estudiaremos para las soluciones presentadas cual es la más eficiente.

1) Multiplicación por sumas sucesivas.
2) División por resta sucesiva.
3) Problema de la potencia por multiplicación sucesiva.
4) Raíz por aproximaciones de newton.
5) Sumatoria por sumas sucesivas.
6) Generación de años bisiestos.
7) Secuencia de fibonacci.
8) El factorial.
9) Equivalente binario de un decimal.
10) Números palíndromos.
11) Numero primos
12) Intercambio de valores entre variables.

Desarrollo del algoritmo de Multiplicación por sumas sucesivas.

Cuado nos topamos con una expresión como 3 * 5 mentalmente damos la respuesta y decimos 15. Esto es porque de una forma abstracta aprendimos a llegar al resultado partiendo de lo que se llama la tabla de multiplicar. La multiplicación es un proceso que tiene una raíz que subyace en una suma sucesiva. Así. Si queremos ver como mediante sumas sucesivas podremos llegar al mismo resultado el proceso a seguir es el siguiente.

La expresión **3 * 5** nos indica que el cinco a de ser sumado a si mismo 3 veces seguidas como se muestra a continuación.

5 + 5 + 5 = 15.

Ahora bien, en una multiplicación existen 3 elementos estos son:

Multiplicando 3
Multiplicador 5
Producto. 15

El multiplicando es el valor que aparece a la izquierda del operador *
El multiplicador es el valor que aparece a la derecha del operador *
El producto es el resultado de efectuar dicha operación.

Por tanto mediante sumas sucesivas el multiplicando indica cuantas veces se ha de sumar a asimismo repetidas veces el multiplicador.

Desarrollo del algoritmo de División por resta sucesiva.

La división es una operación que puede ser efectuada mediante un proceso repetitivo de restas sucesivas. Los elementos que forman parte de una división son:

Dividendo, **Divisor**, **Cociente**, **Residuo**.

10 / **3** = **3** sobrante **1**

Elaborar este proceso mediante restas sucesivas sugiere que al dividendo se le extraiga el divisor tantas veces como sea posible, siempre y cuando el dividendo o lo que va quedando de el sea mayor o igual que el divisor. En este proceso se van contabilizando las veces que esto es posible y al final del proceso se determina que la cuenta de las veces seguidas en que esto ha sucedido viene siendo el cociente y el valor reducido final del dividendo viene a ser el residuo. Veamos:

10 - 3 = 7. como 7 es mayor o igual que tres se hace el proceso restando 3 de 7.
7 - 3 = 4. como 4 es mayor o igual que tres se hace el proceso restando 3 de 4.
4 - 3 = 1. como 1 no es mayor o igual que tres el proceso termina.

Como se han realizado 3 restas sucesivas se concluye que 10 entre 3 es igual a 3. y se tiene como residuo 1.

Desarrollo del algoritmo de la potencia por multiplicación sucesiva.

La potenciación es una operación que consiste en elevar a un número tanta veces como indique su exponente. Así:

$2^2 = 4$, $3^2 = 9$, $5^2 = 25$, $2^3 = 8$, $2^4 = 16$

Mediante multiplicaciones sucesivas esta operación requiere que el número sea multiplicado asimismo tanta veces como indique su exponente así:

$2^2 = 2*2 = 4$, $3^2 = 3*3 = 9$, $5^2 = 5*5 = 25$, $2^3 = 2*2*2 = 8$, $2^4 = 2*2*2*2 = 16$.

Nota: Todo número elevado a la potencia 0 es igual a 1.

Desarrollo del algoritmo de la raíz por aproximaciones de newton.

La radicación es la operación inversa de la potenciación la cual consiste dado un número **x** encontrar uno **y** de tal forma que al multiplicarse por si mismo de como resultado a **x**. veamos:

$\sqrt{4} = 2$ porque $2 * 2 = 4$, $\sqrt{16} = 4$ porque $4 * 4 = 16$

Newton propuso un método para hallar aproximadamente la raíz cuadrada de un número de la siguiente forma.

1. Obtener la parte entera de la mitad del numero x.
2. dividir a x entre dicho resultado y al entero del cociente sumarle dicho resultado para obtener la parte entera de la mitad de dicho resultado.
3. por ultimo repetir el paso 2 dos veces con cada nuevo resultado.

Para entender esto veamos unos cuantos ejemplos.

Ej: 1
X = 9
9 / 2 = 4.5 -> el entero es 4
(9 / 4 + 4) / 2 = (2.5 + 4) / 2 = 6.5 / 2 = 3.25 -> el entero es 3
(9 / 3 + 3) / 2 = (3 + 3) / 2 = 6 / 2 = 3
(9 / 3 + 3) / 2 = (3 + 3) / 2 = 6 / 2 = 3 -> la raíz de 9 = 3.

Ej: 2
X = 49
49 / 2 = 24.5 –> el entero es 24
(49 / 24 + 24) / 2 = (2.04 + 24) / 2 = 26.04 / 2 = 13.02 -> el entero es 13
(49 / 13 + 13) / 2 = (3.76 + 13) / 2 = 13.76 / 2 = 6.88 -> el entero es 6
(49 / 6 + 6) / 2 = (8.16 + 6) / 2 = 14.16 / 2 = 7.08 -> el entero es 7 = raíz 49

Ej: 3
X = 27
27 / 2 = 13.5 -> el entero es 13
(27 / 13 + 13) / 2 = (2.07 + 13) / 2 = 15.07 / 2 = 7.53 -> el entero es 7

(27 / 7 + 7) / 2 = (3.85 + 7) / 2 = 10.85 / 2 = 5.42 -> el entero es 5
(27 / 5 + 5) / 2 = (5.4 + 5) / 2 = 10.4 / 2 = 5.2 -> el entero es 5 = raíz 27

Observe que 27 no tiene raíz exacta pero su aproximación es 5.
Desarrollo del algoritmo de Sumatoria por sumas sucesivas.

La sumatoria por sumas sucesivas es un problema que consiste dado un número x encontrar la sumatoria de todos aquellos números que sean menor o igual que el, así:

Si **x** es 3 la sumatoria de todos los números que son menor o igual que el es 6 ya que

1 + 2 + 3 = 6

si **x** es 10 la sumatoria es 55 ya que

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55.

Desarrollo del algoritmo para determinar si un año es bisiesto.

Según el calendario moderno un año bisiesto ocurre cada 4 años. Se sabe si el año es bisiesto si este cumple con la condición de que sea divisible por 4 y no divisible por 100 o divisible por 400. Esto es año mod 4 = 0 y (año mod 100 <> 0 o año mod 400 = 0). Veamos.

Si el año es 2000

2000 / 4 = 500 , residuo = 0.
2000 / 100 = 20, residuo = 0.
2000 / 400 = 5 , residuo = 0.

Es bisiesto porque es divisible por 4 y a pesar de que es divisible por 100, es divisible por 400.

2004 / 4 = 501, residuo = 0.
2004 / 100 = 20, residuo = 4.
2004 / 400 = 5, residuo = 4.

Es bisiesto porque es divisible por 4 y a pesar de que no es divisible por 400, no es divisible por 100.

1900 / 4 = 475, residuo = 0.
1900 / 100 = 19, residuo = 0.
1900 / 400 = 4, residuo 300.

No es bisiesto porque a pesar de que es divisible por 4, se rechaza porque no puede ser divisible por 100 o si ser divisible por 400.

Investigar el proceso algorítmico de los ejercicios del 7 al 12.

Estructuras de datos.

Cuando se trabaja con variables a cada una de estas en un programa se les asigna un tipo de dato. Con este tipo es que se puede saber cual es la clase de información que dicha variable ha de contener. Si una variable es declarada asignándole un tipo **integer** eso quiere decir que esta solo podrá manejar valores del conjunto de los números enteros. Lo mismo sucede con los tipos **real**, **string**, **char** y **boolean** que son considerados como tipos de datos primitivos cada uno de los cuales representa un conjunto de elementos. Una de las principales limitaciones o "debilidades" que se presenta al trabajar con estos tipos de datos es que permiten la creación de variables que representan valores de datos únicos. Por ej:

A :integer;
Nombre :string;

Estas variables son elementos en un programa que se definen a partir de un tipo simple. A lo largo del programa por ej. Una variable de tipo entero podrá tomar diferentes valores, en un momento determinado solo puede contener un número entero. Si por el contrario en vez de asignar un tipo simple a una variable le asignaremos un tipo estructurado las cosas son diferentes.

Los tipos de datos estructurados pueden ser construidos a partir de tipos simples o de otros tipos estructurados. Estos actúan como un "molde" para variables estructuradas, las cuales pueden contener más de un valor. De esta forma la variedad de los tipos que pueden definirse es enorme.

Una estructura de datos es un conjunto de variables (no necesariamente del mismo tipo) relacionadas entre si de diversas formas.

Trabajando con estructuras de datos es como el programador puede representar los elementos del mundo real, que generalmente son más complejos que un simple número o una letra. Por ej: si es necesario manejar los datos de empleados de una empresa hacerlo a través de múltiples variables seria una tarea ardua y difícil mientras que si se emplea una estructura de datos usando un solo nombre de variable, será posible manipular la información con mucha facilidad.

Operaciones con estructuras de datos.

Las operaciones que se pueden realizar con una estructura de datos son muy distintas a las que se pueden realizar con las variables. Como una estructura de datos es un conjunto el operar sobre sus elementos requiere operaciones tales como:

Recorrido.
La cual consiste en poder tener acceso a cada uno de los elementos de la estructura.

Búsqueda.
La cual consiste en buscar uno o varios de los elementos de la estructura.

Inserción.
La cual consiste en poder añadir elementos a la estructura.

Eliminación.
La cual consiste en poder eliminar elementos de la estructura.

Ordenación.
La cual consiste en poder clasificar u ordenar los elementos de la estructura.

Mezcla.
La cual consiste en poder mezclar los elementos entre dos o más estructuras para formar una nueva estructura.

Una operación muy importante entre estas situaciones lo constituye el **intercambio** que es la operación a través de la cual puede hacerse que un elemento pueda tomar una posición distinta a la actual al intercambiar su posición con otro elemento de la estructura.

Introducción a los arreglos.

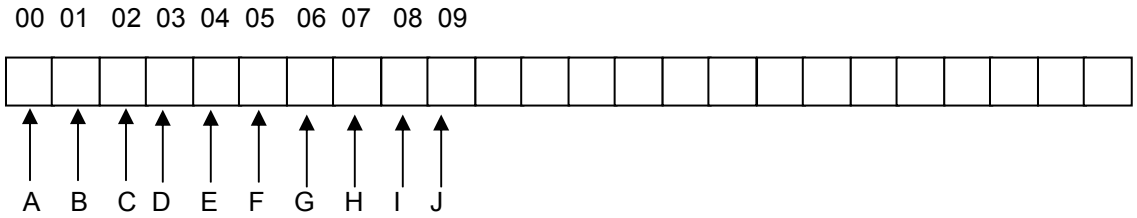
La memoria de la computadora esta compuesta por un conjunto de celdas en las cuales se almacena durante un tiempo la información que estamos manipulando. Cada vez que en un programa utilizamos variables lo que hacemos es reservar en la memoria un subconjunto de estas celdas que pueden ser de 8, 16, ect. Todo esto dependiendo del tamaño en byte del tipo de dato del identificador que estemos utilizando.

Para entender esto veamos lo siguiente.

Vamos a suponer que la siguiente figura representa una porción de la memoria

Cada uno de estos "cuadros" que llamaremos "celdas" representa un byte. Cada vez que creamos una variable lo que hacemos es indicar a la memoria que reserve una o varias de estas celdas. La cantidad de celdas que serán reservadas dependerá del tipo de dato de nuestra variable. Ahora bien:

Los números que vemos ahora sobre cada celda representan para la memoria la dirección o lugar donde se encuentra cada casilla en esta. Si definimos una variable **A** del tipo entero, la asignación o reserva del lugar en la memoria pudiera ser la 00. Si definimos otra, la asignación podría ser la siguiente "01", y a medida que vamos creando variables una nueva celda es reservada para ella. Así, si definimos 10 variables su representación pudiera ser como sigue:



En un programa cuando hacemos esto:

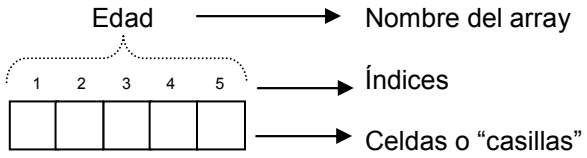
```
A := 10;
B := 35;
C := A+B;
```

Lo que estamos haciendo es colocar en la casilla cuya dirección es "00" el numero 10, en la casilla de B colocamos el numero 35 y en la casilla de C colocamos el numero 45. Observe como las variables actúan como un recipiente en el cual se puede almacenar valores. Ahora

bien, es así como mas o menos sucede en la memoria pero esta situación es tan abstracta que a un programador iniciado no le interesa mucho, él solo se preocupa porque de una manera imaginaria en A hay un 10 en B un 35 y en C un 45.

En la programación en un programa dependiendo del tipo de problema a resolver hay situaciones que se dan en la que aparentemente hay que utilizar una gran cantidad de variables. Hay problemas que para resolverlos de una manera eficiente el utilizar tantas variables de forma independiente no es una buena idea. Es posible agrupar un gran número de celdas y manipularlas bajo un mismo nombre de variable de tal forma que su agrupación represente una estructura organizada de datos. Una de estas estructuras es conocida como un **array**.

Un **array** es un tipo de dato que representa un conjunto cuyos elementos son utilizados empleando un índice. Un array esta estructurado como lo ilustra la siguiente grafica.



Donde:

Nombre del array. Es el nombre de la "variable" que representa al conjunto de celdas.
Índice. Numero entero que representa la posición o numero de cada casilla del array
Celdas. Representa el "recipiente" en el cual se almacena algún valor.

Cuando se define una variable de este tipo, en su declaración siempre se especifica cual es la cantidad de celdas o "casillas" que va a tener. De ahora en adelante para referirnos a las celdas empleáremos el termino "casilla". En Pascal para definir una variable de este tipo se utiliza la palabra reservada array. La declaración puede ser como sigue:

```
Var
    Nombre_array :array[inicio .. fin] of tipo;
```

Donde :

Inicio y **fin** representan un numero entero en el cual inicio normalmente es 1 y fin un numero mayor que uno que indica la cantidad de casillas que ha de tener el array.

Tipo: Es el tipo de dato del array.

Si queremos definir un array de 5 elementos para almacenar números enteros seria como sigue:

```
Var
    Edad :array[1..5] of integer;
```

El nombre de este array es "**edad**"
Tiene **5** elementos o casillas.
Su tipo es **integer**.

Como un array es una estructura que esta formada por un conjunto de elementos llamados "casillas", para poder acceder, utilizar o manipular dichos elementos se procede a escribir el

nombre del array colocando entre corchetes el índice o numero de la casilla que se quiere acceder. Así:

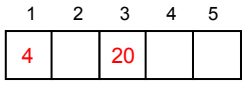
```
Edad[3] := 20;
```

Estamos colocando en la casilla numero **3** el numero 20.

```
Edad[1] := 4;
```

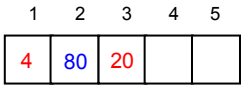
Estamos colocando en la casilla numero **1** el valor 4.

Hasta el momento gráficamente el array estaría como sigue:



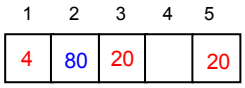

```
Edad[2] := edad[1] * edad[3];
```

Como en la casilla **1** tenemos el valor 4 y en la casilla **3** tenemos el valor 20, en la casilla **2** estamos colocando el valor 80 que es el resultado de multiplicar 4 * 20 o lo que es lo mismo multiplicar el valor de la casilla **1** por el valor de la casilla **3**.




```
edad[5] := edad[5-2];
```

Estamos colocando en la casilla **5** el valor de la casilla **3**, es decir colocando el Numero 20 en la casilla **5**.

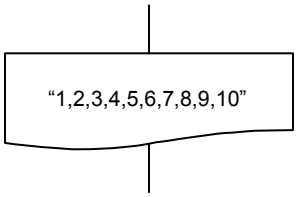


A todas las casillas de un array se le asigna un índice que como ya hemos dicho índice esta vinculado por su posición dentro del array. El conjunto de estos índices normalmente es una secuencia que va desde **1** hasta el número más alto del total de elementos. Si aprendemos a dominar la secuencia de los índices para un array el trabajar con ellos resulta ser una tarea muy fácil. Todos los problemas que podremos tener con array están ligados a la secuencia de dichos índices.

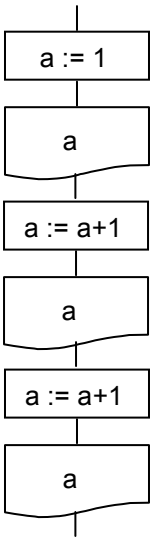
En la unidad numero 2 aprenderemos a trabajar con los array. Mientras tanto vamos a darle un repaso ahora a los bucles que son de vital importancia para trabajar con los arrays.

La repetición de procesos.

Hay problemas de programación en donde ciertos procesos se suceden una y otra vez. Estos tipos de problemas resultan muy tediosos de exponer en un algoritmo si no se toma en cuenta la potencia de lo que es el **ciclo repetitivo**. Intentar resolver un proceso de origen repetitivo de manera puramente secuencial no es la solución mas adecuada. Suponga que deseamos construir un diagrama de flujo que nos imprima los número del 1 al 10. si hacemos lo siguiente:



Es seguro que los números del 1 al 10 se van a imprimir. Ahora bien si lo que deseamos es que se impriman los números del 1 al 1000, entonces esta no es la técnica a utilizar. Imaginase tener que escribir toda una cadena con todos los números del 1 al 1000 para darle salida, es algo tedioso. Una solución aceptable ante esto consiste en generar la secuencia usando un **contador** e ir imprimiendo cada número pero el hacer esto de la siguiente manera:

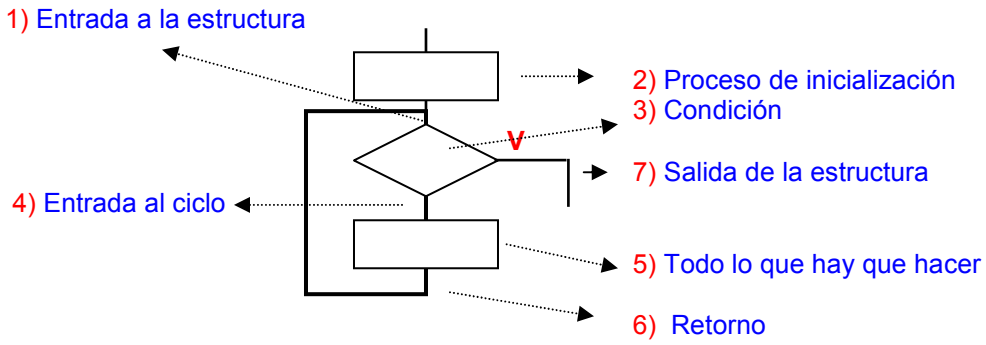


Y continuar sumando un **1** a **a** y darle salida hasta que se haga **1000** veces realmente no es una buena idea. Esta situación se resuelve eficazmente si empleamos lo que se conoce como una estructura cíclica. Así como existe una estructura estándar para hacer preguntas la cual ya conocemos como **IF THEN ELSE**, existe una estándar para hacer que ciertos procesos se repitan tantas veces como sea necesario. Estos tipos de estructuras son conocidos como ciclos hacer mientras o hacer hasta o lo que es lo mismo, ciclos **UNTIL, FOR, REPEAT** y ciclos **WHILE**. Cada uno de estos tiene un comportamiento diferente pero tienen en común que permiten que todos los procesos que se encuentran subordinados a ellos, se hagan una y otra vez. La ventaja de esto es que solo hay que escribir el proceso una sola vez y la misma

estructura se encarga de hacer el resto automáticamente. Lógicamente esto tiene un poder tremendo y nos ayuda a ganar tiempo y espacio a la hora de llevarlo a la práctica. Lo que hay que tener claro es que estas estructuras lo único que hacen es permitir que todo lo que se encuentre dentro de ellas, se procese una y otra vez y que el numero de veces que esto va a suceder esta sujeto siempre a una condición.

Estructura UNTIL.

Esta estructura permite hacer un ciclo hasta que una condición se cumpla. El diagrama de flujo estándar es como sigue:



Entrada a la estructura.
Este es el punto por donde nos conectamos a la estructura. Es el único lugar por donde se puede entrar.

Proceso de inicialización.
Se especifica cual es el valor inicial de la variable que va a comportarse como contador o centinela en la estructura.

Condición.
Aquí se establece la condición sobre el contador o el centinela para que cuando una vez dicha condición se cumpla el ciclo termine y se salga de la estructura.

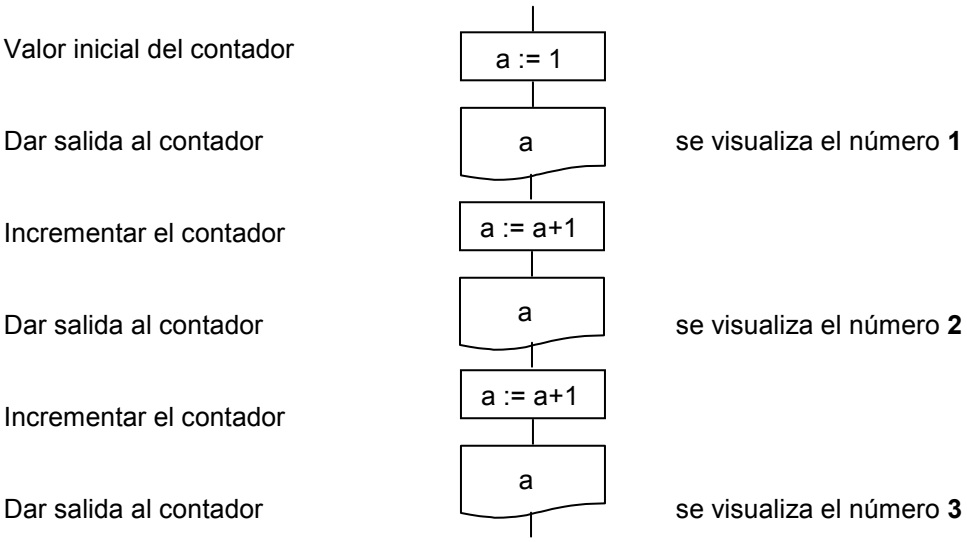
Entrada al ciclo.
Indica que todo lo que viene a continuación esta dentro de la estructura. Es siempre el camino falso de la condición planteada en el diamante.

Todo lo que hay que hacer.
Representa todo el algoritmo que hay que ejecutar una y otra vez.

Retorno.
Especifica el punto donde termina el algoritmo para volver de nuevo al punto de entrada y evaluar nuevamente la condición y verificar si esta al completarse un nuevo ciclo se ha cumplido o no.

Salida de la estructura.
Indica el punto donde la estructura termina una vez la condición se ha cumplido. Es a través de este punto que la estructura se puede conectar con cualquier algoritmo exterior. Observe que hay una **v** a la salida indicando el camino verdadero en el diamante a la condición planteada. El camino falso no hace falta indicarlo ya que se entiende que es la línea que va hacia abajo para entrar al ciclo.

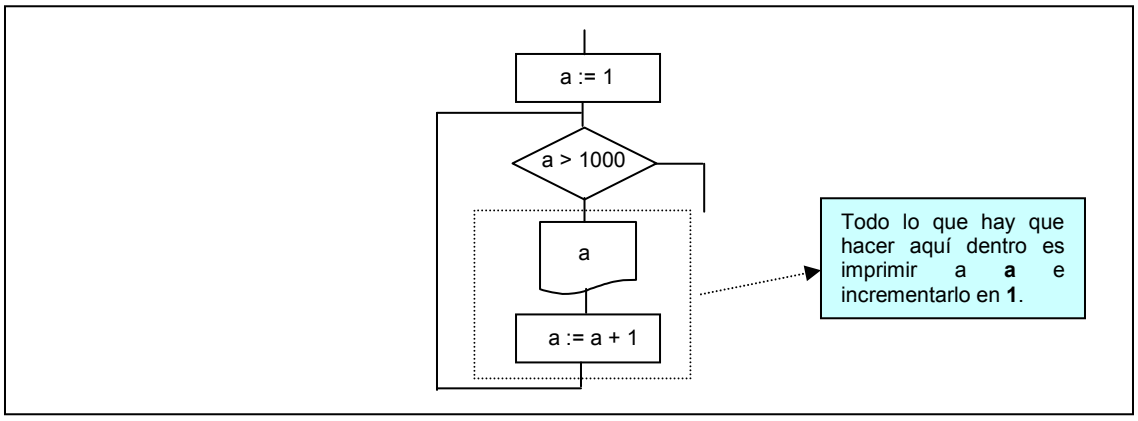
Veamos ahora el problema de los número del **1** al **1000**.



Como se puede dar cuenta hay dos procesos que se repiten una y otra vez. Estos son:

Dar salida al contador
Incrementar el contador

Note que la variable **a** al principio se le da un valor de **1** y luego vemos como se mantiene la generación de la secuencia repitiéndose únicamente la salida de **a** y su incremento. Esta repetición de procesos es a lo que se le llama ciclo debido a que el flujo que llevan dentro del problema es siempre el mismo y durante todo el problema hasta que no se llegue al 1000, estos dos procesos estarán una y otra vez. Ahora bien, como en el problema en ese punto según su flujo hay una repetición continua de esos pasos, esos pasos son los que colocaremos dentro de la estructura cíclica que acabamos de conocer como **UNTIL**. Así utilizando la estructura el diagrama de flujo es como sigue.



Observe: antes de entrar a la estructura la variable **a** se hace igual a **1**. En este caso a será la variable que nos servirá de contador para generar los números del 1 al **1000**. al entrar a la estructura la condición evalúa el valor de **a**, como **a** acaba de entrar su valor es **1**, por tanto la pregunta no se cumple y el flujo sigue hacia abajo y entra a la estructura a desarrollar el

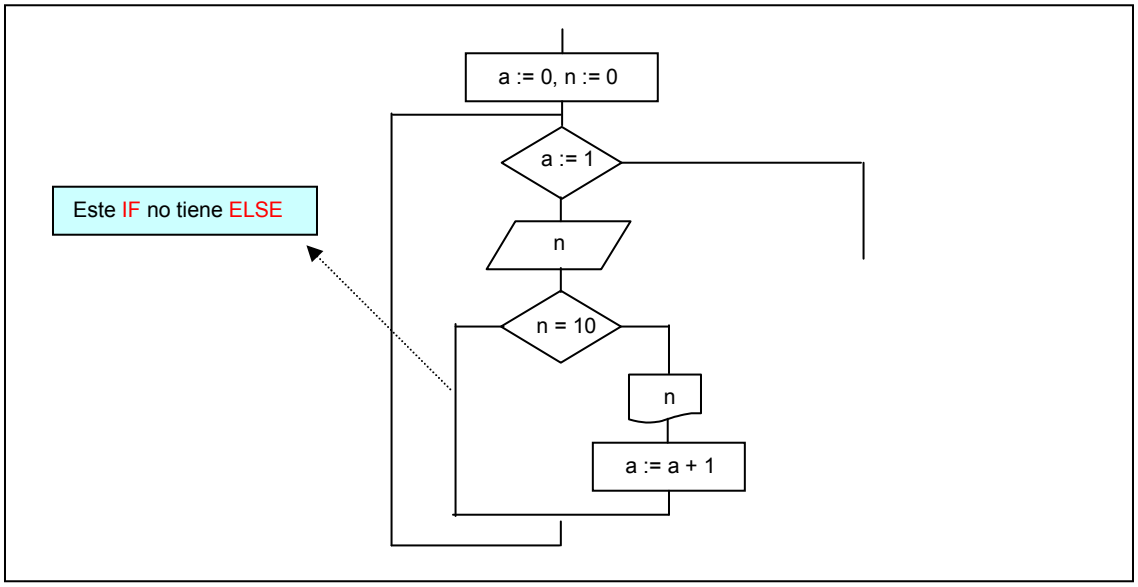
algoritmo que va ha hacer que se imprima la variable **a** y luego se incremente asimismo en **1**. Luego el flujo llega al final del algoritmo y retorna al punto de entrada de la estructura, este retorno se hace cada vez que todo el algoritmo que esta dentro de la estructura se realiza completamente y al retornar vuelve a evaluar la condición esperando que **a** en algún momento llegue a ser > que 1000 y cada vez que un ciclo del algoritmo se complete se estará siempre haciendo lo mismo, ejecutando y evaluando una y otra vez. Note que el valor de **a** siempre se va incrementado ya que cada vez que el proceso **a := a + 1** se ejecuta **a** va creciendo en **1**. Es importante entender que realmente el papel o trabajo de la estructura es hacer que todo el algoritmo que hay dentro de ella sea ejecutado una y otra vez y que cada vez que un ciclo de ejecución de todo el algoritmo se complete, se vuelve a evaluar la condición.

La mayoría de los estudiantes piensan que las estructuras cíclicas son solo para contar y generar secuencias, esto es un error. Las estructuras cíclicas ya sean for, until o while pueden ser usadas para eso pero eso no quiere decir que sean solo para eso. Son herramientas cuyo único objetivo es hacer que todo el algoritmo que se encuentre dentro de ellas se haga una y otra vez, evaluando en cada iteración o desarrollo completo del algoritmo que esta dentro de estas la condición nuevamente.

14) construya un algoritmo que nos permita leer un número por el teclado e imprimirlo solamente cuando este sea el numero 10. El algoritmo se ha de mantener pidiendo el número hasta que sea este igual a 10.

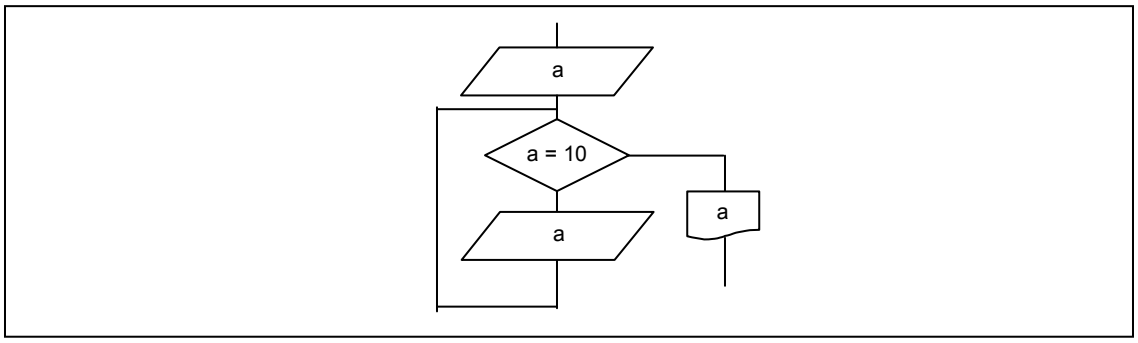
Análisis: Leer un número por el teclado es un algoritmo que ha este punto todos ya sabemos como hacerlo. El problema que ahora se nos presente es que hay que obligar a la lectura que vengan por el teclado a que dicho numero sea el numero **10** y hasta que esto no ocurra estaremos pidiendo el número una y otra vez. Observe que en esto hay un proceso repetitivo o cíclico el cual es leer el número una y otra vez, eso quiere decir que esa lectura estará dentro de nuestro ciclo. Ahora se nos pide que cuando sea el 10 le demos salida o lo imprimamos. El diagrama de flujo es el siguiente.

Si no se usa una buena técnica podría pensarse que este seria el diagrama de flujo.



Es evidente que este diagrama funciona pero no es lo optimo ya que se puede reducir. Segun el enunciado del problema lo que nos interesa es imprimir el numero cuando este sea **10** y mantenerse pidiendo el numero hasta que esto ocurra y ocurrido esto, hemos de terminar. No

siempre es necesario usar un contador para controlar una estructura ciclica, hay situaciones que se dan donde lo que conviene es dejar que un centinela controle la situación. Usando un centinela en este caso el mismo numero que hemos de leer, podemos controlar el numero de veces que la estructura va estar ejecutando el algoritmo que se encuentre dentro de ella. Por tanto el diagrama de flujo quedaria como se muestra a continuación.



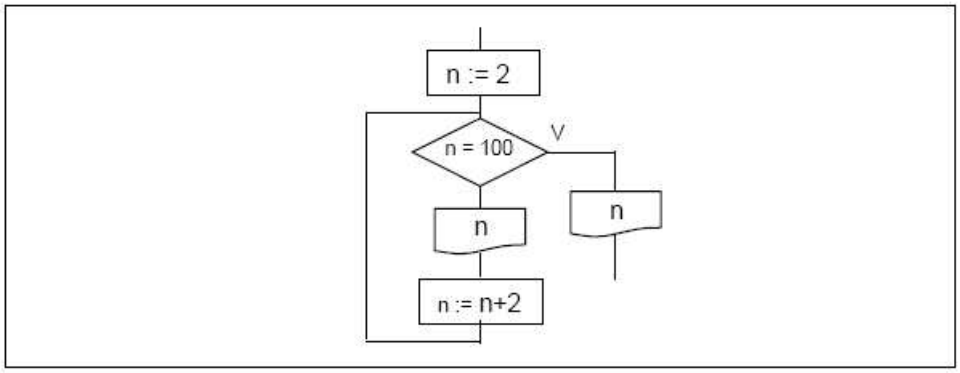
Observe como el numero es pedido una y otra vez y cuando este es **10**, la estructura llega a su fin, se sale y se imprime el valor. Note como antes de entrar a la estructura se ha pedido el numero, si se da la situacion de que en ese momento el numero es 10 sucedera que la pregunta sera cumplida en ese mismo momento y al suceder esto el flujo inmediatamente va hacia afuera, se imprime el numero y se termina. Dese cuenta que ni siquiera ha sido falta usar un contador para controla la estructura, hemos como habiamos dicho usado un centinela. Fijese como la condicion de fin de la estructura en la pregunta del diamente esta sujeta a una entrada del teclado. Es bueno que entienda bien esto, hay controles que lo manejan los contadores, otros los centinelas controlados por asignación y otros centinelas controlados por teclado.

15) haga un diagrama de flujo que nos genera la tabla de multiplicar del 1 al 12 para un numero que se pida por el teclado.

Análisis: la tabla de multiplicar consta de **5 columnas** en la primera esta el valor o numero que indica cual es la tabla, es decir, si es la del 5, la del 7, la del 4, etc. Siempre en esa columna el mismo numero aparecera **12** veces. En la segunda columna aparece el simbolo aritmetico de multiplicar en este caso el asterisco "*****". En la tercera columna lo que aparece es una secuencia de numeros que van desde **1** hasta **12**. en la cuarta columna aparece el simbolo de igualdad y en la quinta el resultado del producto entre la primera columna y la tercera. Toda esta información la encuentra uno al ver una tabla de multiplicar. Algo importante en esto es que hay que hacer la operación **12** veces multiplicando en cada caso por un numero diferente proveniente de una secuencia que va desde **1** hasta **12**. la operación **n * c** donde **n** es la tabla y **c** la secuencia sera realizada una y otra vez **12** veces seguidas. Atendiendo a esto podemos ver que nuestro ciclo tiene que hacerse **12** veces y hemos de colocar dentro de él el algoritmo que realizara la mutiplicación e imprimiró el resultado. Atendiendo a todo esto construya el diagrama de flujo.

16) Haga un diagrama de flujo que genera la progresión aritmética de los números pares hasta el 100.

Una situación en la que no debemos de caer es en la siguiente



Este diagrama de flujo funciona pero no es lo óptimo. Observe que de la manera en que esta organizado el algoritmo ha sido necesario imprimir a **n** al salir del ciclo. Esta situación a sido provocada porque en la condición se está planteando la pregunta que si **n** ya es **100** se salga, pero al hacer esto el valor **100** que también es parte de la secuencia no se le dará salida dentro del bucle ya que al cumplirse la condición el ciclo termina y es necesario al salir imprimir nuevamente a **n** para que el valor **100** sea reflejado. Ahora bien, si nos fijamos bien este paso innecesario viene por la forma en que se esta haciendo la pregunta en el diamante, esa situación seria mas perfecta si en vez de preguntar que si es igual a **100** preguntáramos si es mayor que **100** para que el valor **100** se imprima dentro del ciclo y al incrementarse nuevamente a **n** tomara el valor **102** y al evaluarse la condición nuevamente el ciclo se romperá y no será necesario tener fuera del ciclo la salida o la impresión de **n**.

A medida que vamos conociendo mas herramientas para trabajar podremos en esa misma proporción crear diagramas más complejos. Vamos a ver diagramas ahora en los que los problemas tengan que involucrar todo lo que hasta ahora hemos visto.

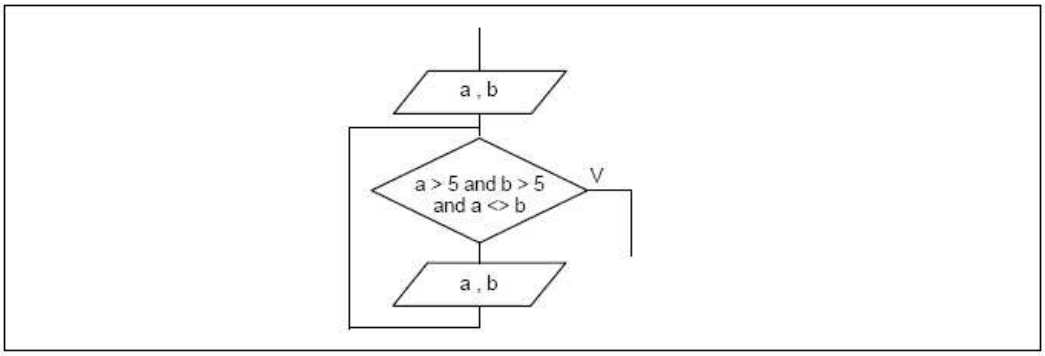
17) Construya un diagrama de flujo que pida 2 números por el teclado que sean mayor o igual que **5** y sean diferentes entre si, si el primero es mayor que el segundo genere la progresión geométrica de los números naturales que comienzan a partir del segundo numero leído y terminan en el cuadrado del primer numero leído sin imprimir en la progresión aquellos números que sean múltiplos del entero de la raíz del segundo. Si el primero numero leído no es el mayor pida un tercer numero que sea diferente a los dos primeros y mayor que el segundo y calcule la media aritmética de la progresión geométrica que hay desde el primer numero hasta el tercero sin tomar en cuenta en la progresión el numero que sea igual al segundo.

Análisis: Este problema es complejo a la vez que interesante y para poder resolverlo hay que tener muy claro que es lo que se esta pidiendo. Por tanto en casos como estos hay que tener cuidado a la hora de empezar a resolverlo sin antes haber echo un buen análisis y haber descubierto todo su flujo. Lo primero que haremos será escribir un algoritmo para indicar de manera general la secuencia de los pasos que hemos de seguir para llegar a la solución.

a) Pedir 2 números por el teclado
b) Verificar que ambos cumplan con la condición de ser mayor o igual que 5 y diferentes entre si y hasta que esto no ocurra volver a pedir los números.
c) Comparar ambos números
    Si el primero es mayor
        Generar la progresión aritmética comenzando a partir del segundo sin
        Hacer que se impriman aquellos números que sean múltiplos del entero de la raíz del segundo terminando la progresión en el cuadrado del primer numero leído.
    Si el primero no es mayor
        Pedir un tercer numero con la condición de que este sea diferente a los dos primeros y mayor que el segundo.
        Generar la progresión geométrica que va a partir del primer número leído hasta el tercero sin considerar el número en la progresión geométrica que sea igual al segundo.
        Con todos los valores de la progresión calcular la media aritmética.

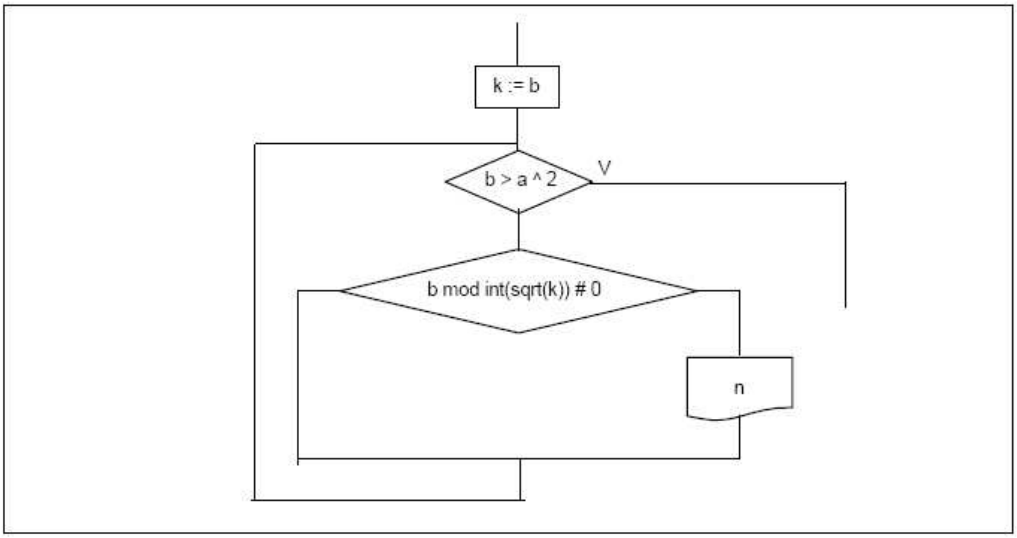
En cierta forma podría pensarse que este algoritmo es lo mismo que hemos dicho en el enunciado, es decir hemos escrito lo mismo otra vez. Esto es cierto pero, es seguro que haciendo esto empezamos a tener una idea mas clara de cual es la secuencia en que lrán los pasos para ir resolviendo el problema. Esta técnica hemos de emplearla muchas veces en la programación y la diagramación hasta que no seamos capaces ya, de haber desarrollado la habilidad y la intuición para hacer las cosas. Claro si el problema es sumamente grande es de esperarse que esta forma de enfocar las cosas sea de mucha ayuda.

El punto b si observamos bien tiene en su flujo un ciclo en el cual también se involucra el punto a. esto nos dice que la captura de los 2 números hay que hacerla usando un ciclo y dicho ciclo va a terminar cuando la condición de que ambos sean mayor que 5 y diferentes entre si se cumpla. Por el momento vamos a desarrollar esta parte.



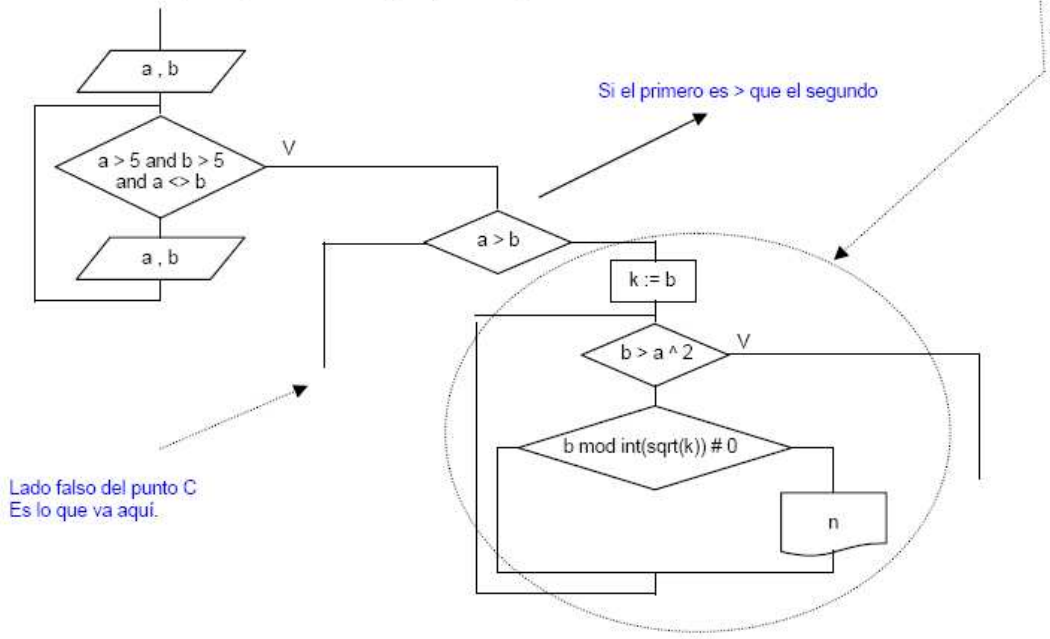
Observe como usando el operador lógico **and** nos hemos ahorrado el tener que haber hecho un **if** para cada condición.

Ahora vamos a construir la parte que corresponde a la progresión aritmética del lado verdadero del punto c.



La razón de haber usado la variable **k** ha sido porque **b** dentro del ciclo es usado como contador para generar la secuencia a partir de si mismo y como el ciclo tiene que generar la progresión hasta que **b** sea > que el cuadrado del primer numero, **b** ira perdiendo su valor. **k** es usado como una copia inicial de **b** para que se pueda mantener la pregunta de que la secuencia que es el **if** que esta dentro del ciclo no imprima aquellos valores cuyo modulo respecto del valor inicial de **b** que es el segundo numero sea diferente de cero o lo que es lo mismo no sean múltiplo de **k** que es el valor inicial de **b**.

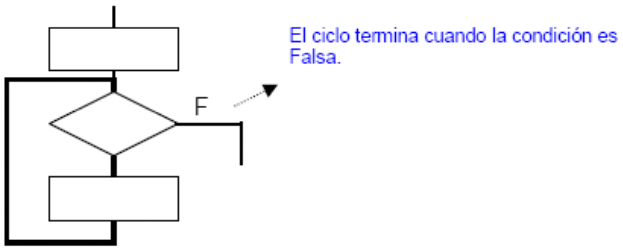
Ahora vamos a ensamblar esto de tal manera que nos quede en la parte verdadera del **if** que ha de preguntar, una vez los **2** números leídos cumplen con la condición de ser mayor que **5** y diferentes entre si, si el primero es mayor que el segundo.



Construya la parte del diagrama que corresponde al lado falso del punto c y ensamble todo el diagrama.

La Estructura tipo **WHILE**.

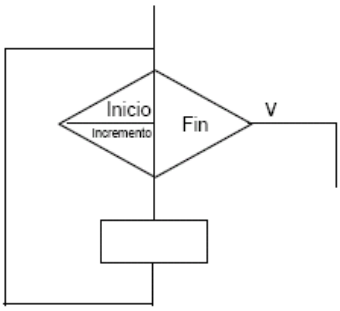
Esta estructura al igual que la **UNTIL** es usada también para hacer que todo lo que hay dentro de ella se haga una y otra vez. A diferencia de un UNTIL el **While** hace que el ciclo de repeticiones sea posible mientras una condición se este cumpliendo. En le caso del **UNTIL** es lo contrario pues en este ocurre hasta que se cumpla una condición. El diagrama de flujo de esa estructura es como sigue.



Note que el diagrama de flujo la única diferencia que tiene con el del **UNTIL** es que aquí se pone una **F** y en el **UNTIL** se pone una **V**.

La Estructura For.

Esta estructura es parecida a la estructura **Until.** En esta el contador de la estructura viene definido en la misma sintaxis. Los bucles **for** son buenos utilizarlos cuando necesitamos hacer procesos conociendo de antemano que el número de veces que el ciclo se va a ejecutar esta predeterminado directamente por el contador. Si la situación de parada esta sujeta a una bandera o a una entrada del teclado entonces estos no son recomendables usarlos aunque, existen instrucciones que detienen abruptamente las iteraciones aunque esto no es una buena técnica. Los bucles **for** permiten hacer ciclos hasta que una condición sea cumplida. Un diagrama de flujo para este tipo de estructura puede interpretarse como sigue:



La forma de trabajar del bucle **for** consiste en especificar cual es valor inicial de la variable contador " **inicio** " luego, se especifica el **incremento** y se indica hasta donde debe llegar el contador contabilizando las iteraciones. Una vez la condición de fin se cumple el bucle sale como se muestra en el diagrama. Todo lo que esta simbolizado por el cuadro es todo el algoritmo que se ha de ejecutar una y otra vez hasta que se cumpla la condición de fin. Hay otros diseños de esta estructura que en vez de un diamante utilizan un rectángulo.

Tarea:

1) Investigar en la Internet los bucles anidados o anillos y traer ejemplos.