Programación

Unidad 5: Estructuras de repetición

"La repetición es la madre de la habilidad".

~ Tony Robbins

Estructuras de Repetición

Hasta ahora en nuestros algoritmos cada una de las instrucciones se ejecutaba como máximo una vez.

Esto nos obliga a repetir instrucciones cuando queremos hacer más de una vez la misma operatoria.

Por ejemplo supongamos que tenemos el algoritmo que indica si una persona es mayor de edad.

```
Proceso Mayor
    Definir edad como Entero;
    Mostrar “Ingrese la edad de la persona: “;
    Leer edad;
    Si ( edad >= 18) Entonces
        Mostrar “Es mayor de edad”
    Sino
        Mostrar “Es menor de edad“;
    Fin Si
FinProceso
```

Si ahora deseamos hacer lo mismo pero para 3 personas, deberíamos triplicar todas las acciones referidas a tomar la edad y evaluarla.

```
Proceso Mayor
    Definir edad como Entero;
    Mostrar “Ingrese la edad de la persona: “;
    Leer edad;
    Si ( edad >= 18) Entonces
        Mostrar “Es mayor de edad”
    Sino
        Mostrar “Es menor de edad“;
    Fin Si

    Mostrar “Ingrese la edad de la persona: “;
    Leer edad;
    Si ( edad >= 18) Entonces
        Mostrar “Es mayor de edad”
    Sino
        Mostrar “Es menor de edad“;
    Fin Si

    Mostrar “Ingrese la edad de la persona: “;
    Leer edad;
    Si ( edad >= 18) Entonces
        Mostrar “Es mayor de edad”
    Sino
        Mostrar “Es menor de edad“;
    Fin Si

FinProceso


```

Pero...¿Y si ahora necesitamos hacerlo para 10 personas? Tendríamos que replicar las instrucciones de la línea 3 a la línea 9 **diez veces**. Lo cual hace que nuestro algoritmo sume cada vez más líneas, aunque no resuelva problemas más complejos.

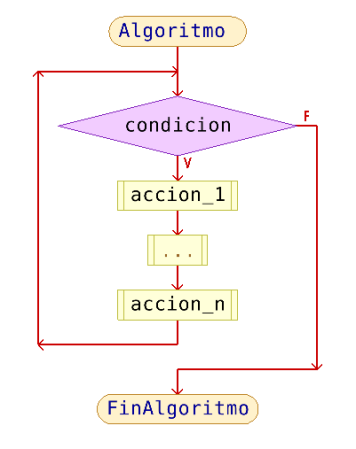
Así surge la necesidad de poder indicar en un algoritmo que ciertas instrucciones deben ejecutarse más de una vez.

Las estructuras que utilizaremos para esto las denominaremos estructuras iterativas, o estructuras de repetición, o ciclos de repetición. El objetivo de estas estructuras es generar ciclos iterativos en los que ciertas instrucciones se repitan una cierta cantidad de veces.

Estructura de repetición "Mientras...Hacer (While)"

La primer estructura iterativa que utilizaremos es el Mientras (While), esta estructura repetirá un bloque de instrucciones **mientras** se cumpla una cierta condición.

Pseudocódigo	Diagrama de Flujo	Codificación C/C++

Mientras (condición) **Hacer**
  acción_1;
  ....
  acción_N;
FinMientras
		**while** (condicion) {
		  statement_1;
		  ....
		  statement_n;
		}

¿Cómo funciona esta estructura?

Cuando dentro de un programa se alcanza una sentencia "Mientras", se evalúa la *condición* que se encuentra entre paréntesis.

- Si el resultado de esta evaluación es Verdadero (true) entonces "entra al bucle" y se ejecutarán la o las acciones que se encuentre dentro de este bloque
- Al finalizar con estas instrucciones (terminar una iteración) se volverá a evaluar la condición.

- Cuando el resultado de evaluar la condición es Falso (false) entonces no se ejecuta la iteración.

Cabe aclarar que podría suceder que nunca se ingrese al bucle si la condición resulta Falso en la primera vez que se evalúa.

Para generar una cantidad de iteraciones, debemos pensar **qué condición** generará esa cantidad de "vueltas" dentro del ciclo de repetición. Un ejemplo es crear una variable **contador** inicializada en 0 e ir incrementando su valor dentro del ciclo.

Veamos el ejemplo anterior realizado con esta estructura:

```
Proceso Mayor
    Definir edad como Entero;
    Definir contador como Entero;
    contador = 0;

    Mientras (contador < 10) Hacer
       Mostrar "Ingrese la edad de la
       persona: ";
       Leer edad;
       Si ( edad >= 18) Entonces
          Mostrar "Es mayor de edad";
       Sino
          Mostrar "Es menor de edad";
       Fin Si
       contador = contador + 1;
    FinMientras

FinProceso
```

```
int main() {
    int edad;
    int contador;
    contador = 0;

    while (contador < 10) {
        cout << "Ingrese la edad";
        cin >> edad;
        if (edad >= 18) {
            cout << "Es mayor de edad";
        }
        else {
            cout << "Es mayor de edad";
        }
        contador = contador + 1;
    }
    return 0;
}
```

En esta solución deberemos ingresar sí o sí 10 edades, y en caso que el usuario deseara ingresar más o menos edades deberemos hacer un nuevo algoritmo.

Una forma de hacer que nuestro algoritmo sea más flexible y le permita al usuario ingresar N edades es cambiar la condición del Mientras, y en lugar de usar un valor fijo de 10 utilizar uno variable que también pueda ser ingresado por el usuario:

```
Proceso Mayor
    Definir edad como Entero;
    Definir cantidad como Entero;
    Definir contador como Entero;
    contador = 0;

    Mostrar "¿Cuántas edades ingresará?";
    Leer cantidad;

    Mientras (contador < cantidad) Hacer
```

```
int main() {
    int edad;
    int contador;
    int cantidad;
    contador = 0;

    cout <<  "¿Cuántas edades ingresará?";
    cin >> cantidad;
    while (contador < cantidad) {
        cout << "Ingrese la edad";
```

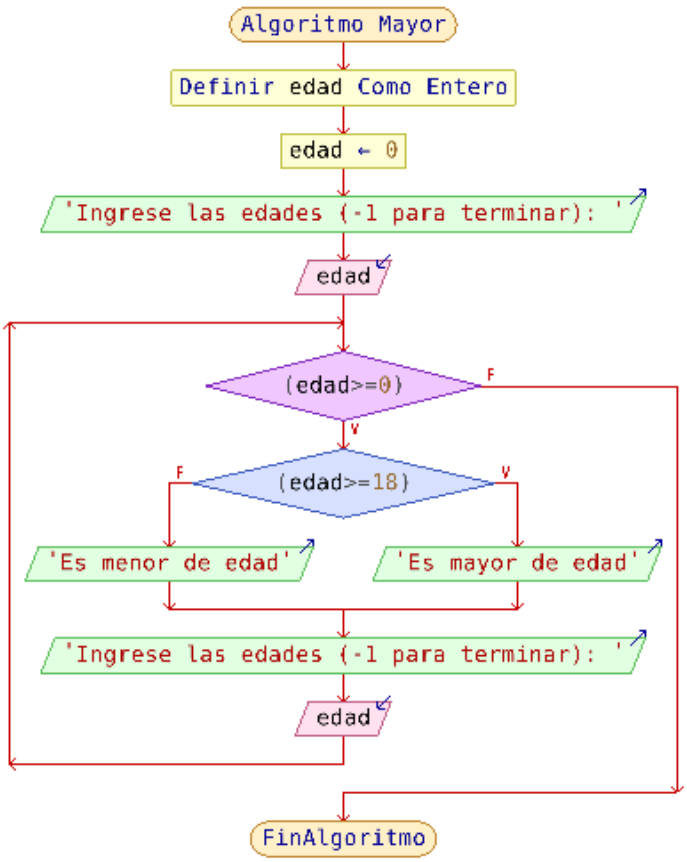
```
11          Mostrar “Ingrese la edad de la
12          persona: “;
13          Leer edad;
14          Si ( edad >= 18) Entonces
15             Mostrar “Es mayor de edad”;
16          Sino
17             Mostrar “Es menor de edad“;
18          Fin Si
19          contador = contador + 1;
20       FinMientras
21
22    FinProceso
```

```
        cin >> edad;
        if (edad >= 18) {
            cout << “Es mayor de edad”;
        }
        else {
            cout << “Es mayor de edad”;
        }
        contador = contador + 1;
    }
    return 0;
}
```

Ahora nuestro algoritmo permitirá ingresar N cantidad de edades.

Otra forma de resolverlo hubiese sido que el usuario ingresara un valor negativo para indicar que quiere terminar la carga (dado que no hay edades negativas).

El diagrama de esta solución sería:



A diferencia de la solución anterior, aquí no tenemos necesidad de contar ya que no estamos buscando generar una cantidad de iteraciones conocidas, sino que el ciclo se cortará al ingresar un valor negativo.

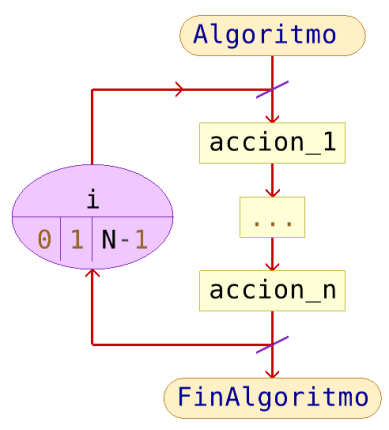
Algo a tener en cuenta es que debo tomar el primer valor fuera del ciclo para no terminar utilizando valores de corte en las instrucciones. En lugar de incrementar un contador como últimas instrucciones dentro del ciclo, lo que tendremos es el nuevo ingreso de una edad.

Esta estructura es de las que llamamos **No exactas** porque la cantidad de iteraciones podría variar en la ejecución.

Estructura de repetición “Para...Hacer (For)”

La estructura Para funciona de manera similar al Mientras pero establece dentro de la misma estructura una variable que contará las iteraciones. Por esta razón decimos que el Para es una estructura de iteración de repeticiones **Exactas.**

Pseudocódigo	Diagrama de Flujo	Codificación C/C++

Para i<-0 **Hasta** N-1 **Hacer**
  accion_1
  ...
  accion_n
FinPara
		**for** (int i = 0; i < N; i++) {
		  statement_1;
		
		  ....
		
		  statement_n;
		}

¿Cómo funciona esta estructura?

En el ciclo Para se utiliza lo que se llama la **variable del índice**. Esta variable tiene que estar definida como una variable de tipo Entero. En nuestro ejemplo, la variable de control es **i**. La variable de control se inicializa con un número (en este caso 0) e irá tomando todos los valores enteros hasta llegar al número **N -1**

¿Cómo se realiza esta asignación?

Luego de inicializarse (en su valor inicial), se evalúa si ya se alcanzó el valor final, en caso que aún no lo haya alcanzado se ejecuta el cuerpo del ciclo que se compone de 1 o más instrucciones (bloque).

Luego de ejecutar el cuerpo del ciclo, se incrementa en uno el valor de **i**, es decir, que internamente se realiza una asignación **i = i + 1**

Luego, el ciclo "Para" evalúa el valor de **i** y si todavía no llegó a **N - 1** (el valor final) vuelve a ejecutar el cuerpo del ciclo. Y así sucesivamente hasta que **i** alcanza su valor final.

Es importante destacar que el incremento de la variable del índice no se debe realizar en el algoritmo, ya que forma parte del funcionamiento de la estructura del "Para". Esta es una diferencia con respecto a los otros ciclos en donde debemos asegurarnos que la condición se modifique para poder finalizar.

También es importante que comprendas que los valores iniciales y finales son también válidos, es decir que la primera iteración se ejecutará con **i = 0** , y la última iteración se ejecutará con **i = N - 1**

Como el Para es un ciclo de repeticiones exactas, necesitaré saber de antemano cuántas iteraciones quiero generar, ya sea porque el usuario lo ingresó, o porque es una cantidad conocida

Rehagamos el ejemplo anterior con un ciclo Para

```
Proceso Mayor
    Definir edad como Entero;
    Definir cantidad como Entero;

    Mostrar "¿Cuántas edades ingresará?";
    Leer cantidad;
    Para i<-0 Hasta cantidad - 1 Hacer
       Mostrar "Ingrese una edad: ";
       Leer edad;
       Si ( edad >= 18) Entonces
          Mostrar "Es mayor de edad";
       Sino
          Mostrar "Es menor de edad";
       Fin Si
    FinPara

FinProceso
```

```
int main() {
    int edad;
    int contador;

    cout <<  "¿Cuántas edades ingresará?";
    cin >> cantidad;
    for (int i = 0; i < cantidad; i++) {
        cout << "Ingrese una edad";
        cin >> edad;
        if (edad >= 18) {
            cout << "Es mayor de edad";
        }
        else {
            cout << "Es mayor de edad";
        }
    }
    return 0;
}
```

Veamos paso a paso cómo se ejecuta el algoritmo:

Paso	cantidad	edad	i	Mensaje	Comentario
5	-	-	-	-	Se imprime el mensaje solicitando una cantidad de edades
6	**3**	-	-	-	Se lee la cantidad ingresada
7	3	-	0	-	Se define e inicializa la variable i Se controla que se cumpla la condición del Para
8	3	-	0	-	Se imprime el mensaje pidiendo una edad
9	3	**23**	0	-	Se lee la edad ingresada
10	3	23	0	-	Evaluar la condición edad >= 18
11	3	23	0	**“Es mayor”**	Por ser verdadera la evaluación se imprime el mensaje “Es mayor de edad”
12	3	23	**1**	-	Finaliza la primer iteración Incrementa el contador en 1
13	3	23	1	-	Vuelve a evaluar la condición del Para(i < cantidad) Como se cumple, sigue en el bucle y comienza la 2da iteración
8’	3	23	1	-	Se imprime el mensaje pidiendo una edad
9’	3	**12**	1	-	Se lee la edad ingresada
10’	3	12	1	-	Evaluar la condición edad >= 18
11’	3	12	1	**“Es menor”**	Por ser falsa la evaluación se imprime el mensaje “Es menor de edad”
12’	3	12	**2**	-	Finaliza la segunda iteración Incrementa el contador en 1
13’	3	12	2	-	Vuelve a evaluar la condición del Para(i < cantidad) Como se cumple, sigue en el bucle y comienza la 3era iteración
8’’	3	12	2	-	Se imprime el mensaje pidiendo una edad
9’’	3	**54**	2	-	Se lee la edad ingresada
10’’	3	54	2	-	Evaluar la condición edad >= 18
11’’	3	54	2	**“Es mayor”**	Por ser verdadera la evaluación se imprime el mensaje “Es mayor de edad”
12’’	3	54	**3**	-	Finaliza la tercer iteración Incrementa el contador en 1
13’’	3	54	**3**	-	Vuelve a evaluar la condición del Para(i < cantidad) Como no se cumple, porque ambos valen 3 sale del bucle
14	3	54	**-**	-	Finaliza la ejecución (retorna 0)

Cambiando el incremento

Como se detalla en la descripción paso a paso de cómo se modifica el contador del **Para**, al finalizar cada iteración se incrementa el valor de la variable índice (la i en nuestro caso).

En los ejemplos que vimos el incremento siempre fue de 1 es decir que i toma el valor i + 1, o de otra forma i = i + 1 (que es lo mismo que i++ en c). Este valor en que se incrementa la variable contador es llamado **paso** o **incremento** del Para, y puede modificarse.

Veamos un ejemplo, si desearamos generar los números pares 2, 4, 6, 8, 10 y 12 utilizando un ciclo de repetición podríamos generar todos los números desde 2 al 12 (el 0 y 1 los salteamos porque no nos interesan) y por cada número preguntamos si es par o no.

```
1  Proceso Pares
2      Para i<-2 Hasta 12 Hacer
3          Si ( i % 2 == 0) Entonces
4             Mostrar “Es par:”, i;
5          Fin Si
6      FinPara
7  FinProceso
8
```

```
int main() {
    for (int i = 2; i <= 12; i++) {
        if (i % 2 == 0) {
            cout << “Es par:” << i;
        }
    }
    return 0;
}
```

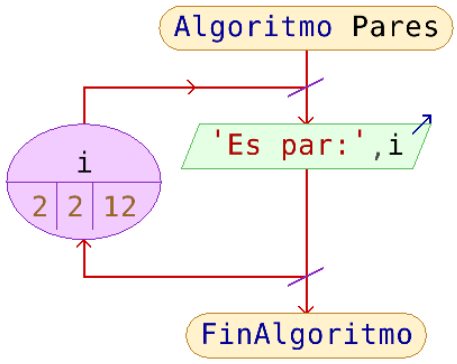
La anterior es una solución completamente válida, y generará los números deseados. Pero modificando el paso o incremento existe una solución mucho más sencilla en la que no es necesario preguntar si el valor del contador es par o no. Dado que sabemos que si el contador arranca en 2, si el paso fuera de 2 también entonces todos los números generados serían pares, por lo tanto no es necesario el condicional y podemos eliminarlo.

```
1  Proceso Pares
2      Para i<-2 Hasta 12 Con Paso 2 Hacer
3          Mostrar “Es par:”, i;
4      FinPara
5  FinProceso
6
```

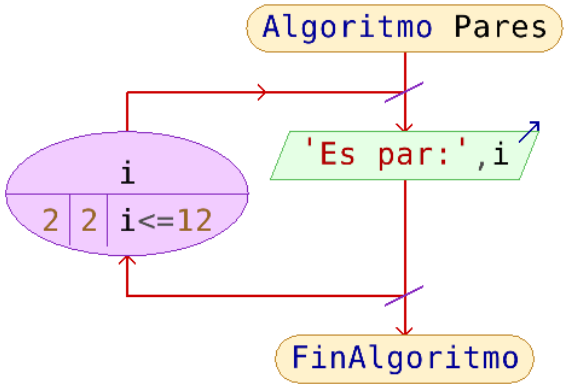
```
int main() {
    for (int i = 2; i <= 12; i = i + 2) {
        cout << “Es par:” << i;
    }
    return 0;
}
```

En el diagrama aunque no lo hayamos aclarado anteriormente, existe un lugar específico donde se indica el incremento o paso, y es la posición del medio de la burbuja. Anteriormente no le prestamos atención pero si se mira el diagrama del ejemplo se verá que en el medio se encuentra el valor 1, ya que estábamos usando incrementos de 1.

Veamos cómo quedaría el diagrama del algoritmo de Pares mejorado:



Si quisiéramos usar la misma condición que en el código c/c++ el diagrama quedaría de la siguiente forma:



En lugar de el valor final, colocamos la condición de corte. Ambos diagramas son equivalentes. Recuerde que si hay un valor final, este es **inclusive**

Anidando estructuras de repetición

Al igual que con las estructuras de control, las estructuras de repetición pueden contener entre sus acciones otras estructuras.

Si revisamos algunos de los ejemplos vistos hasta ahora veremos que dentro de las acciones a realizar en cada iteración había una estructura condicional, recordemos siempre que las estructuras establecen bloques de código y estos pueden anidarse indefinidamente en caso de ser necesario.

Lo particular de anidar estructuras de repetición es que generará una cantidad iteraciones de la estructura anidada por cada iteración de la estructura de repetición principal.

Veámoslo con un ejemplo, supongamos que deseamos calcular el promedio de notas de todo el curso. Sabiendo que tenemos 30 alumnos y 4 notas por alumno, el algoritmo resultaría:

```
Proceso Promedios
    Definir nota como Entero;
    Definir prom como Real;
    Definir nombre como Cadena
    Para i<-0 Hasta 29 Hacer
        Mostrar “Ingrese nombre”;
        Leer nombre;
        prom = 0;     // Inicializa prom
        Para j<-0 Hasta 3 Hacer
           Mostrar “Ingrese nota”;
           Leer nota;
           prom = prom + nota;  //acumula
        FinPara
        prom = prom / 4;  //Calcua el prom
        Mostrar “Promedio:”, prom;
    FinPara
FinProceso
```

```
int main() {
    int nota;
    float prom;
    string nombre;
    for (int i = 0; i < 30; i++) {
        cout << “Ingrese nombre:”;
        cin >> nombre;
        prom = 0;
        for (int j = 0; j < 4; j++) {
            cout << “Ingrese una nota”;
            cin >> nota;
            prom = prom + nota;
        }
        prom = prom / 4;
        cout << “Promedio:” << prom;
    }
    return 0;
}
```

El primer Para, generará 30 iteraciones (una para cada alumno) dado que el contador i tomará los valores 0, 1, 2, … , 29. Mientras que el segundo Para genera 4 iteraciones (una por cada nota del alumno) dado que el contado j tomará los valores 0, 1, 2, 3.

Al estar el Para de las notas anidado en el el Para de los alumnos resulta que para cada iteración de alumno, se generan 4 iteraciones de notas, es decir, por cada alumno se solicita el ingreso de 4 notas. Que es lo que queríamos lograr.

El anidamiento de estructuras no está limitado a los Para obviamente, sino que se pueden anidar ciclos Mientras, o combinar un ciclo Mientras con uno de tipo Para o viceversa, **todo dependerá del problema a solucionar**.

Veamos un ejemplo donde se combina un Mientras con un Para: Supongamos que, siguiendo el mismo ejemplo anterior, el curso no necesariamente es de 30 alumnos sino que el Profesor al realizar la carga decide terminarla ingresando el nombre “Fin”.

Recordemos que por no saber a priori cuántos alumnos se desean cargar no podremos usar un ciclo exacto como el Para, sino que tendremos que utilizar un Mientras para generar las iteraciones de cada alumno, sin embargo si sabemos que son 4 notas por cada alumno por lo cual podríamos usar un Para como estructura de repetición para pedir las notas.

```
Proceso Promedios
    Definir nota como Entero;
    Definir prom como Real;
    Definir nombre como Cadena;
    Mostrar “Ingrese nombre”;
    Leer nombre;
    Mientras nombre != “Fin” Hacer
        prom = 0;     // Inicializa prom
        Para j<-0 Hasta 3 Hacer
           Mostrar “Ingrese nota”;
           Leer nota;
           prom = prom + nota;  //acumula
        FinPara
        prom = prom / 4;  //Calcua el prom
        Mostrar “Promedio:”, prom;

        Mostrar “Ingrese nombre”;
        Leer nombre;
    FinMientras
FinProceso
```

```
int main() {
    int nota;
    float prom;
    string nombre;
    cout << “Ingrese nombre:”;
    cin >> nombre;
    while (nombre != “Fin”) {
        prom = 0;
        for (int j = 0; j < 4; j++) {
            cout << “Ingrese una nota”;
            cin >> nota;
            prom = prom + nota;
        }
        prom = prom / 4;
        cout << “Promedio:” << prom << endl;

        cout << “Ingrese nombre:”;
        cin >> nombre;
    }
    return 0;
}
```

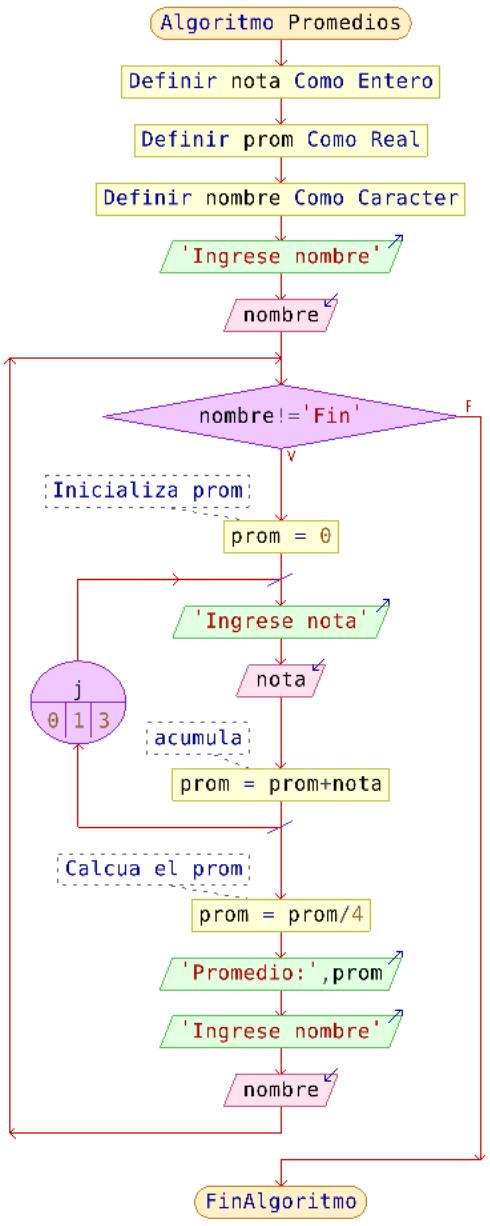
En los ejemplos que hemos visto del Mientras la condición de corte siempre estaba vinculada a una variable contador que se actualizaba dentro del ciclo. Sin embargo en este ejemplo vemos que la condición no es un contador ya que no sabemos qué cantidad de iteraciones debemos generar, en este caso la condición de corte evalúa el valor ingresado por el usuario.

De igual forma que antes se inicializaba el contador en 0 antes del Mientras, ahora debemos tomar un valor ingresado por el usuario antes del Mientas para poder evaluarlo. Luego las últimas acciones de cada iteración serán volver a pedir el valor.

El resultado de este algoritmo es que se pedirán nuevos alumnos y sus 4 notas mientras que nose ingrese como nombre “Fin”, con lo cual tenemos un ciclo inexacto para los alumnos (ya no

sabemos con seguridad cuántos alumnos se ingresarán) y un ciclo exacto anidado que permite ingresar las 4 notas del alumno (esto sigue como en el ejemplo anterior).

Veamos el diagrama del algoritmo anterior:



TODO: Otros Bucles menos usados: Hacer...Mientras y Hacer...Hasta

Bibliografía utilizada

Luis Joyanes Aguilar, Ignacio Zahonero Martínez. Programación en C. Segunda Edición. España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA. S.A.U., 2005. ISBN-84: 481-9844-1.

● Brian W. Kernighan, Rob Pike. La práctica de la programación. Pearson Educación. Méjico (2000)

● Debugging with CodeBlocks: http://wiki.codeblocks.org/index.php?title=Debugging_with_Code::Blocks