Universidad Nororiental Privada **Gran Mariscal de Ayacucho**
Facultad de Ingeniería
Escuela de Ingeniería
Escuela de Ingeniería en Informática e Ingeniería de Sistemas
Núcleo Ciudad Guayana

ASIGNATURA: PROGRAMACION II
SECCION: 3T1
LAPSO ACADEMICO: II-2019
PROF (A): YELENIA BOADA R.

GUIA DE ESTUDIO
Tema: ESTRUCTURAS REPETITIVAS O CONTROLADORA DE FLUJO DE INFORMACION

1. BUCLES O CICLOS

Un bucle, lazo, ciclo o *loop* (en inglés) es un segmento de algoritmo o programa (una secuencia de instrucciones) que se repiten un determinado número de veces mientras se cumple una determinada condición, en otras palabras, un bucle o ciclo es un conjunto de instrucciones que se repiten mientras una condición es verdadera o existe.

A cada repetición del conjunto de acciones se denomina **iteración**. Para que un bucle no se repita indefinidamente debe tener una condición de **parada o de fin**. Esta **condición de parada** o de fin se verifica cada vez que se hace una iteración. El ciclo o *loop* llega a su fin cuando la condición de parada se hace verdadera

La condición de parada puede estar al principio de la estructura repetitiva o al final.

Al igual que las estructuras de selección simple o compuesta (los bloques si – entonces – sino – fin si), en un algoritmo pueden utilizarse varios ciclos. Estos **ciclos pueden ser independientes** (una a continuación de otro) **o anidados** (ciclos dentro de ciclos).

¿Para qué me sirve esto?

Los ciclos o procesos repetitivos de instrucciones son procedimientos fundamentales en el uso de las computadoras y por lo tanto en muchos algoritmos y programas.

Vamos a utilizar ciclos cuando:

Necesitemos **REPETIR INSTRUCCIONES** un determinado número de veces, mientras se cumpla una condición, mientras un hecho sea verdadero o hasta cuando se alcance un determinado valor o condición.

Cuando necesitemos **CONTAR** un determinado número de elementos o de acciones, por ejemplo contar las sílabas de un texto, los elementos de una secuencia que verifican una determinada condición o contar el número de personas que cumplen ciertas características. En este caso se incluirán **contadores** dentro del bucle.

Los **contadores son variables** (generalmente de tipo **Entero**) que tienen un valor inicial y que se incrementan o decrementan en un valor constante cada vez que ocurre una iteración. Cuando los contadores se decrementan se habla de descontar, en lugar de contar.

También usaremos ciclos cuando necesitemos **ACUMULAR** o **TOTALIZAR** terminados valores cada vez que se realiza una iteración. Los **acumuladores también son variables** (generalmente de tipo **Entero**, **Real** o **String**), que almacenan valores variables resultantes de las operaciones que se realizan en cada ciclo.

Por ejemplo, podemos usar para ir sumando los precios de varios vehículos y luego calcular el precio promedio general (con una variable acumulador de tipo **real**), para calcular la potencia o el factorial de un número a través de multiplicaciones sucesivas (un acumulador de tipo **entero** o **real**) o para ir agregando a una cadena de caracteres, letras o sílabas que construirán un mensaje (una variable acumulador del tipo **string**)

2. Estructura Iterativa Para … (*For …*)

Es una estructura iterativa que es controlada por una variable (llamada también **variable índice**), la cual se incrementa o decrementa hasta llegar a un valor límite o valor final que representa la condición de parada.

La estructura **Para** comienzan con un valor inicial de la variable índice, las acciones especificadas para el ciclo se ejecutan un número determinado de veces, a menos, que el valor inicial de la variable índice sea mayor que el valor límite que se quiere alcanzar.

SE RECOMIENDA USARLO: la estructura **Para** es recomendada cuando se conoce el número de veces que se deben ejecutar las instrucciones del ciclo, es decir, en los casos en que el número de iteraciones es fijo y conocido.

El incremento o decremento de la variable_índice suele ser de 1 en 1, salvo cuando se indica lo contrario. La variable índice suele ser de tipo **Entero** y se utilizan comúnmente nombres como i, j o k (no importa si en mayúsculas o minúsculas)

Sintaxis
for (<inicio;final;contador>)

Ejemplo:

```
for(int i = 0;i <= 10;i++)
    cout <<”hola”;
```

3. Estructura iterativa Hacer – Hasta (Do – *While*)

Ejecuta un bloque de instrucciones varias veces hasta que se cumple la condición que es verificada al final del bucle.
Las instrucciones dentro del ciclo **Repetir** se van a realizar mientras la condición de parada evaluada al final sea falsa. Dicho de otro modo, el ciclo se va a detener cuando la condición de parada se haga verdadera.

SE RECOMIENDA USARLO: la estructura **Repetir** es recomendada cuando las instrucciones del ciclo se pueden realizar **al menos 1 vez antes** de comprobar la condición de parada.

Sintaxis

```
Do {
 Sentencias
}while (<condicion>);
```

Ejemplo

```
do {
      cout <<"entre la nota";
      cin>>nota; i++;
} while(i <=10);
```

4. Estructura iterativa Mientras – (While …)

Es una estructura iterativa que permite verificar la condición de **entrada** al ciclo **antes** del cuerpo de instrucciones a repetir.

Como la evaluación de la condición de entrada se realiza al **inicio** del bloque **Mientras**, puede ocurrir que las instrucciones del ciclo no se realicen ni siquiera 1 vez, a diferencia del **Repetir**, donde el bloque de instrucciones se realiza al menos 1 vez porque la condición de parada se verifica al final. Las instrucciones del Mientras se pueden realizar 0 o más veces antes de que se cumpla la condición de terminar el ciclo.

El conjunto de instrucciones dentro del **Mientras** se ejecuta cuando la condición de entrada del principio se cumple (es verdadera). Dicho de otro modo, el ciclo de instrucciones dentro del **Mientras** se va a detener cuando la condición se haga falsa.

SE RECOMIENDA USARLO: la estructura **Mientras** es recomendada cuando tienes que verificar la condición de entrada al inicio y si se cumple, entonces, entrar al ciclo y realizar sus instrucciones.

Sintaxis

while (<condición>)

Ejemplo:

```
while (a <= 10){
   a++;
   suma+=a;
}
```

¿CUÁL USAR? CON EJEMPLOS:

1. *Si la condición de parada del ciclo depende de que hayas repetido 30 veces el ciclo:*

Usas un bloque **Para** con la sintaxis: Para i = 1 hasta 30 en 1 hacer
No necesitas una instrucción dentro del **Para** que incremente el valor de la variable i, ya que la instrucción "en 1" del encabezado del **Para** se encarga de incrementar y actualizar esta variable índice.

2. *Si la condición de parada depende de que hayas repetido 30 veces el ciclo o de que leas la palabra "detener":*

No deberías usar un **Para**, porque no necesariamente vas a repetir el ciclo 30 veces. Si el usuario te suministra en alguna repetición del ciclo la palabra "detener" saldrías del ciclo aunque no lo hayas repetido 30 veces.

Además en la sintaxis del **Para** NO PUEDES verificar que la palabra leída sea la "detener"

En el Para **NO PUEDES COLOCAR** algo como Para i = 1 (hasta 30 o palabra ="detener") en 1 hacer
Ÿ
Usas un bloque **Repetir** o un bloque **Mientras** que **sí te permiten verificar en una expresión** las condiciones de que se haya repetido 30 veces el ciclo o que la palabra suministrada sea "detener"

Cuando uses un Mientras o un Repetir, debes incluir instrucciones dentro del ciclo que actualicen las variables de la condición de parada, para este ejemplo, debería tener una instrucción que aumente el valor de i (i = i + 1) y otra que te permita leer

un nuevo valor para la variable palabra (Escribir(“¿desea detenerse?”) Leer(palabra)).

ejemplos

1. Calcular el promedio de un alumno que tiene 7 calificaciones

```
int main()
{
float cal,suma=0,pro;
cout<<"Digite sus 7 calificaciones)";
for (int i=7;i>0;i--)
{
cin>>cal;
cout<<" ";
suma+=cal;
}
pro=suma/7;
cout<<"El promedio de las calificaciones es " <<pro
getch();
}
```

2. Leer 10 números y obtener su cubo y su cuarta.

```
Int  main()
{
float n,cubo,cuar;
int y=6,b=6,q=6;
cout<<"Digite 10 numeros)";
cout<<"NUMERO CUBO CUARTA";
for (int i=10;i>0;i--)
{
cin>>n;
cout<<" ";
cubo=n*n*n;
cuar=n*n*n*n;
cout<<n:
cout<<**;
cout<<y++;
b++;
q++;
}
getch();
}
```