

# Programación – Sentencias de control – Estructuras Repetitivas



Departamento de Informática - Facultad Politécnica

# Objetivos

- Comprender conceptos asociados a las estructuras repetitivas.
- Aprender a usar instrucciones de repetición while, do-while y for.
- Aprender a usar instrucciones de saltos en ciclos.
- Implementar el funcionamiento de la instrucción goto.

# Contenido de la presentación

- Necesidad de estructuras de repetición
- Bucle while
- Bucle do-while
- Bucle for
- Uso de banderas
- Uso de las instrucciones break y continue
- Etiqueta: goto
- Uso de exit

# Estructuras repetitivas

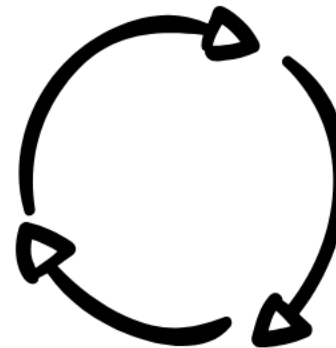
- Hay una gran variedad de situaciones que requieren que una o varias instrucciones se repitan varias veces, ya sean cálculos u otro tipo de instrucciones. Las estructuras repetitivas abren la posibilidad de realizar una secuencia de instrucciones más de una vez.

Ejemplo: calcular el promedio de calificaciones de 5 (o  $N$ ) alumnos de una clase.

Puntaje	Calificación
Entre 91 y 100	5
Entre 81 y 90	4
Entre 71 y 80	3
Entre 60 y 70	2
Menos de 60	1

# Estructuras repetitivas

- Un bucle, lazo o loop es una sección de código que se repite. Es decir cuando se termina de ejecutar la última instrucción del conjunto, el flujo de control retorna a la primera sentencia y comienza una nueva repetición de las sentencias que forman esa sección de código.
- Se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones, la iteración se asocia a un número entero que indica el número de veces que se repite un trozo de código.
  - Mientras (**while**)
  - Hacer mientras (**do-while**)
  - Desde hasta/Para (**for**)

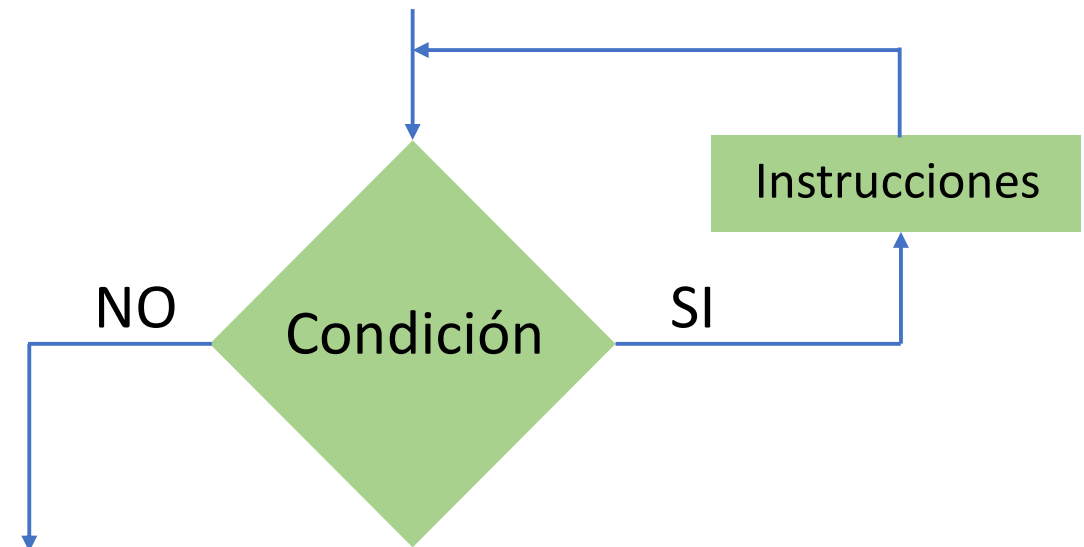


# Bucle while

- Es el tipo de bucle más sencillo. Sintaxis:

```
while(expresión){  
    .....  
    sentencias;  
    .....  
}
```

- El bucle while comienza por evaluar la expresión. Si es cierta, se ejecuta las sentencias. Luego se vuelve a evaluar la expresión. De nuevo, si es verdadera, se vuelve a ejecutar las sentencias. Este proceso continúa hasta que el resultado de evaluar la expresión es falso. Por esto se le llama a esta expresión la condición de salida.



# Bucle while

- Normalmente, en las sentencias del bucle while se coloca alguna instrucción que modifique la expresión de control. Lo más habitual es utilizar un bloque de sentencias en vez de una sentencia única. Por ejemplo:

```
int variable=10;
while(variable > 1){
    printf("la variable vale %d \n", variable);
    variable= variable-1; //modifica la expresión de control
    printf("valor tras decrementar la variable %d\n", variable);
}
```

Un bloque es un conjunto de instrucciones limitadas por llaves **{ }**. Si se desea ejecutar más de una instrucción en una estructura de control (selectiva o repetitiva) **se debe utilizar un bloque.**

# Bucle while

```
#include <stdio.h>

int main(){
    int variable= 10;
    while(variable > 1){
        printf("la variable vale %d \n", variable);
        variable= variable - 1;
        printf("valor tras decrementar la variable %d\n", variable);
    }
    return 0;
}
```

C:\Clases\cc\2020\_ProgramacionNC\Ej\_while\_1\bin\Debug\Ej\_while\_1.exe

```
la variable vale 10
valor tras decrementar la variable 9
la variable vale 9
valor tras decrementar la variable 8
la variable vale 8
valor tras decrementar la variable 7
la variable vale 7
valor tras decrementar la variable 6
la variable vale 6
valor tras decrementar la variable 5
la variable vale 5
valor tras decrementar la variable 4
la variable vale 4
valor tras decrementar la variable 3
la variable vale 3
valor tras decrementar la variable 2
la variable vale 2
valor tras decrementar la variable 1
```

```
Process returned 0 (0x0)   execution time : 0.102 s
Press any key to continue.
```



# Acumuladores y Contadores

## Contador:

- Es una variable que se incrementará en una unidad cada vez que se ejecute el proceso.
- El contador se utiliza para llevar la cuenta de determinadas acciones durante la ejecución del programa.
- Un contador debe ser inicializado, lo cual consiste en asignarle un valor inicial (generalmente cero).

`contador = contador+1;`      ó    `contador+=1;`      ó    `contador++;`

## Acumulador:

- La principal diferencia con el contador es que el incremento (o decremento) de cada suma es variable en lugar de constante.
- Puede ser un acumulador aditivo o multiplicativo.
- Un acumulador también debe ser inicializado (0 para la suma y 1 para la multiplicación).

`acumulador = acumulador + x;`      ó    `acumulador+= x;`

# Bucle while

## Ejemplo 1

Dado un número natural  $n$ , desarrolle un algoritmo que calcule la sumatoria y el promedio de los números menores a  $n$  y que sean múltiplos de 3.

```
#include<stdio.h>
int main(){
    int n,i=1,sum=0,cont=0;
    float prom;
    printf("\nIngrese el numero: ");
    scanf("%d",&n);
    while(i<n){
        if((i%3)==0){sum+=i;cont++;}
        i++;
    }
    if(cont){
        prom=1.0*sum/cont;
        printf("\nLa suma es: %d",sum);
        printf("\nEl promedio es: %.3f",prom);
    }
    else{
        printf("\nEl nro ingresado es menor que 4");
    }
    return 0;
}
```

# Bucle while

**Ejemplo 2:** Desarrolle un algoritmo que permita determinar a partir de un número de días (ingresado por pantalla), los años, meses, semanas y días que constituyen el número de días proporcionado.

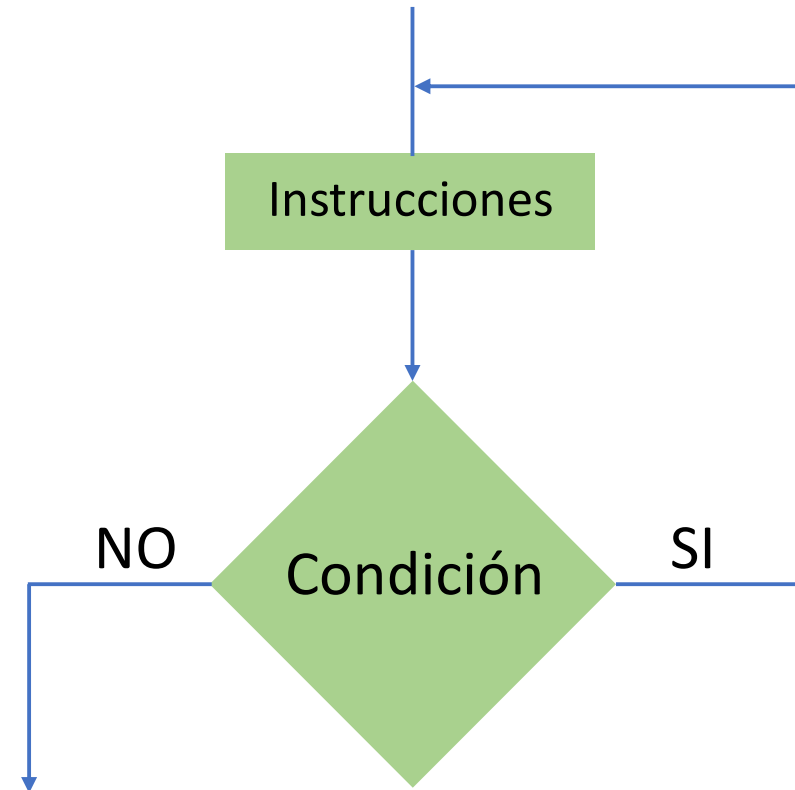
```
#include<stdio.h>
int main(){
    int dias,meses=0,anhos=0,semanas=0;
    printf("Ingrese el numero de dias: ");
    scanf("%d",&dias);
    while(dias>=365){
        dias-=365;
        anhos++;
    }
    while(dias>=30){
        dias-=30;
        meses++;
    }
    while(dias>=7){
        dias-=7;
        semanas++;
    }
    printf("\nLa cantidad de anhos es: %d",anhos);
    printf("\nLa cantidad de meses es: %d",meses);
    printf("\nLa cantidad de semanas es:
%d",semanas);
    printf("\nLa cantidad de dias es: %d",dias);
    return 0;
}
```

# Bucle do - while

La sintaxis de este bucle es:

```
do{  
    .....  
    instrucción;  
    .....  
while(expresión);
```

Su funcionamiento es análogo al del bucle `while`, salvo que la expresión de control se evalúa al final del bucle. Esto nos garantiza que el bucle `do-while` se ejecuta al menos una vez. Es menos habitual que el bucle `while`.



# Bucle do-while

Uso común: validación de datos

## Ejemplo 1: – Ingresar un número positivo

```
#include<stdio.h>
int main(){
    int n;
    do{
        printf("\nIngresa un numero positivo: ");
        scanf("%d",&n);
    }while(n <= 0);
    return 0;
}
```

# Bucle do-while

**Ejemplo 2:** Calcular el factorial de un número  $n$ .

Nota: usaremos un acumulador multiplicativo.

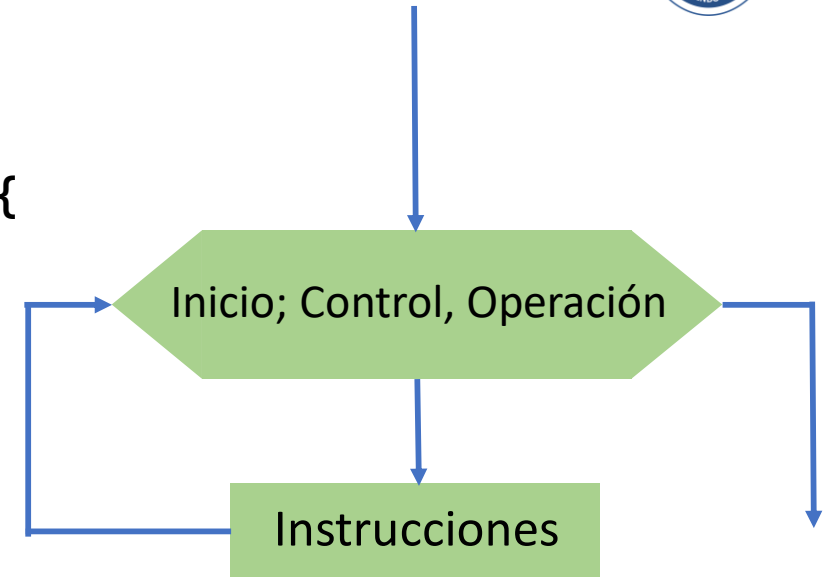
```
#include<stdio.h>
int main(){
    int n, fact=1, i=1;
    printf("\nIngresa el nro n: ");
    scanf("%d",&n);
    do{
        fact*= i;
        i++;
    }while(i <= n);
    printf("\nEl factorial de %d es: %d",n,fact);
    return 0;
}
```

# Bucle for

La sintaxis del bucle for es:

```
for (variable_control= valor_inicio; condicion; operación){  
    .....  
    sentencias;  
    .....  
}
```

Uno de los usos comunes de este bucle consiste en realizar una acción un número determinado de veces. Está compuesto de tres expresiones: la de **inicio**, la de **control** y la de **operación** al final de una iteración; y de una instrucción (o instrucciones).



Un ejemplo sencillo de uso es:

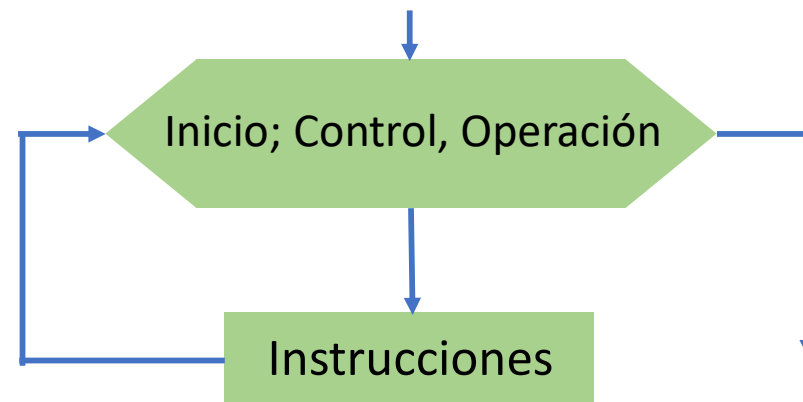
```
for (i=0; i<10; i++){  
    printf( "i vale %d \n", i);  
}
```

Esta versión del bucle imprime un mensaje en la pantalla mientras que no se alcance la condición de salida,  $i < 10$ .<sup>15</sup>

# Bucle for

El funcionamiento del bucle for es el siguiente:

1. Primero se ejecuta la expresión de **inicio**. Normalmente esta es una expresión de asignación a una variable, que le da un valor inicial.
2. Luego se comprueba la expresión de **control**. Si esta expresión es verdadera se ejecuta la sentencia, o el grupo de sentencias. Si la expresión es falsa el bucle finaliza.
3. Tras ejecutarse la sentencia (o sentencias) se evalúa la expresión de **operación**. Habitualmente lo que hace esta expresión es incrementar la variable de control.
4. A continuación se vuelve al segundo paso. El bucle finaliza cuando la expresión de control es falsa.





# Bucle for

En un bucle for podemos omitir la expresión de **inicio**; por ejemplo, si sabemos que la variable ya está inicializada:

```
int i=0;
for ( ; i<10; i++)
    printf("i vale %d\n", i);
```

También podemos omitir la expresión de **operación**. Esto es habitual cuando la variable de control ya se modifica dentro del bucle. Por ejemplo:

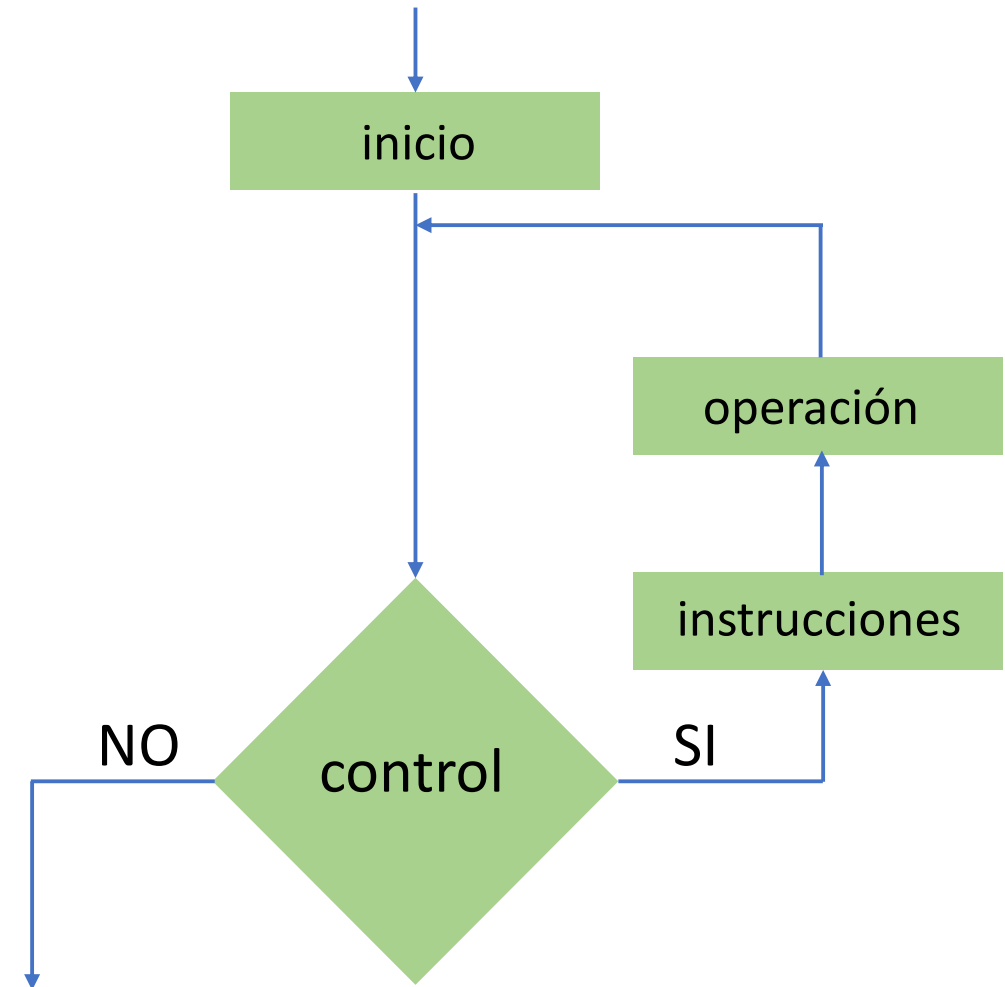
```
int i=0;
for( ; i<10; ){
    printf("i vale %d\n",i);
    i++;
}
```

# Bucle for y while

El bucle for es equivalente a un bucle while escrito del siguiente modo:

```
inicio;  
while(control){  
    instrucciones  
    operación;  
}
```

El bucle for se sigue el orden: evaluación de control, ejecución de instrucciones y evaluación de operación.



# Bucle for

**Ejemplo 1** - Extraído de <http://c2.com/cgi/wiki?FizzBuzzTest>

Imprimir en pantalla los números comprendidos entre 1 y 100. Pero para los múltiplos de 3, imprimir “Fizz” en lugar del número, mientras que para los múltiplos de 5 se imprime “Buzz” en lugar del número. Si el número es múltiplo de 3 y de 5, mostrar “Fizzbuzz” en lugar del número.

```
#include<stdio.h>
int main(){
    int i;
    printf("\nNumeros del 1 al 100");
    for(i=1;i<101;i++){
        if((i%3==0)&&(i%5==0)) printf("\nFizzBuzz");
        else if(i%3==0) printf("\nFizz");
        else if(i%5==0) printf("\nBuzz");
        else printf("\n%d",i);
    }
    return 0;
}
```

# Bucle for

**Ejemplo 2** – Imprimir todas las letras minúsculas en el alfabeto (inglés) de la tabla ASCII.

Nota: recordar que las variables de tipo char almacenan números enteros.

```
#include<stdio.h>
int main(){
    char minuscula; //sera un contador!!
    printf("Las letras minusculas son:\n");
    for(minuscula='a'; minuscula<='z';minuscula++){
        printf("%c ",minuscula);
    }
    printf("\n");
    return 0;
}
```

 C:\Clases\cc\2020\_ProgramacionNC\Ej\_For\_2\bin\Debug\Ej\_For\_2.exe

```
Las letras minusculas son:
a b c d e f g h i j k l m n o p q r s t u v w x y z

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

# Uso de Banderas o Interruptores

Una **bandera** o **interruptor**, es una variable que puede tomar los valores falso (cero) o verdadero (distinto de cero) a lo largo de la ejecución de un programa, comunicando así información de una parte a otra del mismo. Pueden ser utilizados para el control de bucles.

## Ejemplo:

```
int contador=0;
int bandera=0 ;
while(bandera==0){
    contador = contador + 1 ;
    if(contador ==10){
        bandera=1;
    }
}
```

# Uso de Banderas o Interruptores

**Ejemplo 3** – Escribir un programa que determine si un número ingresado por teclado es un número primo o no.

```
#include<stdio.h>
int main(){
    int n,k;
    printf("Ingrese el numero: ");
    scanf("%d",&n);
    int esPrimo=1; //bandera
    for(k=2;k<n;k++){
        if(n%k==0){ //k es divisor de i?
            esPrimo=0; //Si lo es, entonces no es primo
        }
    }
    if(esPrimo) printf("El numero %d es primo.\n",n);
    else printf("El numero %d no es primo.\n",n);
    return 0;
}
```

# Uso de Break

En lenguaje C, para escribir una instrucción de salto **break** (interrumpir), se utiliza la sintaxis:

```
break;
```

La instrucción de salto break se usa para interrumpir (romper) la ejecución normal de un bucle. Así, el control del programa se transfiere (salta) a la primera instrucción después del bucle.

```
while(1){ //Siempre se queda dentro del while
    ---
    ---
    if(condición){
        break; //sale del while
    }
    ---
    ---
}
```

# Uso de Break

**Ejemplo 3-2** – Escribir un programa que determine si un número ingresado por teclado es un número primo o no.

```
#include<stdio.h>
int main(){
    int n,k;
    printf("Ingrese el numero: ");
    scanf("%d",&n);
    int esPrimo=1; //bandera
    for(k=2;k<n;k++){
        if(n%k==0){ //k es divisor de i?
            esPrimo=0; //Si lo es, entonces no es primo
            break; //Sale del bucle, ya se encontro un divisor!
        }
    }
    if(esPrimo) printf("El numero %d es primo.\n",n);
    else printf("El numero %d no es primo.\n",n);
    return 0;
}
```



# Uso de Continue

En lenguaje C, para escribir una instrucción de salto **continue** (continuar), se utiliza la sintaxis:

```
continue;
```

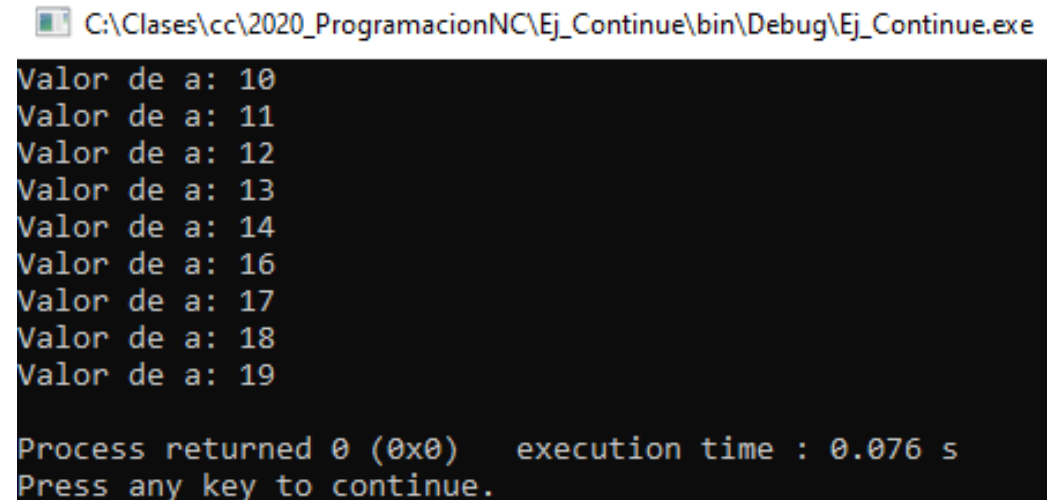
Esta instrucción se usa para interrumpir (romper) la ejecución normal de un bucle. Sin embargo, el control del programa no se transfiere a la primera instrucción después del bucle (como **break**); sino que finaliza la **iteración en curso**, transfiriéndose el control del programa a la condición de salida del bucle, para decidir si se debe realizar una nueva iteración o no.

```
while(condición){  
    ---  
    ---  
    if(condición salto){  
        continue; // no se ejecuta el resto de la iteración  
    }  
    ---  
    ---  
}
```

# Uso de Continue

En este ejemplo se imprimen los valores del 10 al 20, excepto el 15.

```
#include<stdio.h>
int main(){
    int a=10;
    do{
        if( a == 15) {
            // Salta esta iteración.
            a++;
            continue; //Vuelve al inicio del bucle!
        }
        printf("Valor de a: %d\n",a);
        a++;
    }while(a<20);
    return 0;
}
```

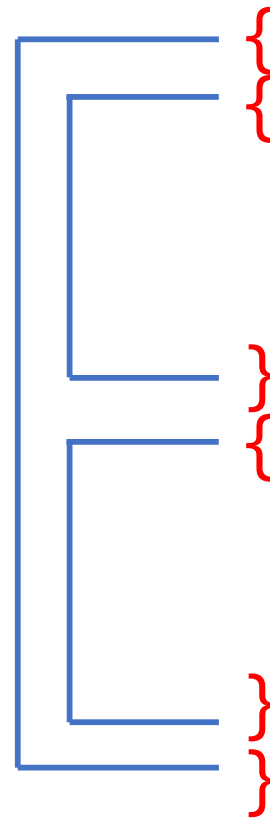
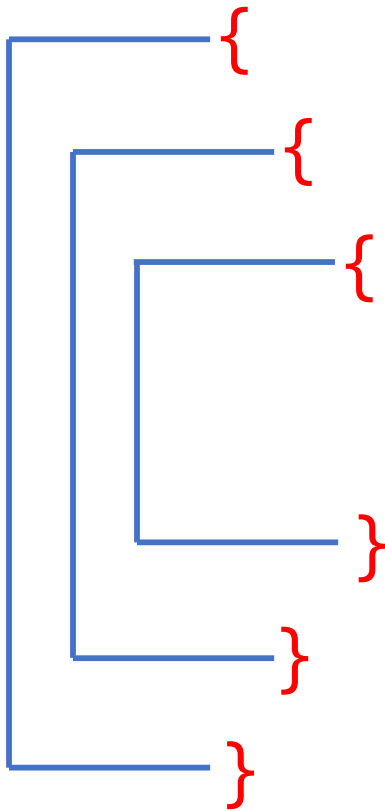


```
C:\Clases\cc\2020_ProgramacionNC\Ej_Continue\bin\Debug\Ej_Continue.exe
Valor de a: 10
Valor de a: 11
Valor de a: 12
Valor de a: 13
Valor de a: 14
Valor de a: 16
Valor de a: 17
Valor de a: 18
Valor de a: 19

Process returned 0 (0x0)   execution time : 0.076 s
Press any key to continue.
```

# Anidamientos de Bucles

(Bucles en cascada)



**Observación:** al poner el cursor al lado de una llave en el IDE (CodeBlocks y DevC++ ), nos muestra la llave correspondiente en el bloque

# Uso de etiquetas: instrucción goto

Una etiqueta se define mediante su nombre (identificador) seguido del carácter dos puntos (:).

ETIQUETA: ... (instrucción)

**goto** Es una instrucción de salto incondicional dentro del ámbito de una función, y nos permite dar un salto a la parte del programa donde se encuentre la etiqueta respectiva. ***Un programa debidamente estructurado debe evitar la utilización del goto, de hecho es la instrucción prohibida en la programación estructurada.***

Su sintaxis es:

```
goto ETIQUETA;
```

Cuando se ejecute la instrucción goto, el programa saltará y continuará su ejecución a partir de la etiqueta marcada.

# Uso de etiquetas: instrucción goto

Qué hace el siguiente ejemplo?

```
#include <stdio.h>
int main () {
    int inicio=1, fin=10;
    CICLO: printf("Ahora imprimo %d\n",inicio);
    inicio++;
    if(inicio<=fin)
        goto CICLO;
    printf("Fin del programa\n");
    return 0;
}
```

Como se puede observar, **goto** puede usarse para crear un bucle, salir de bucles anidados, o para ir a una parte del código u otra si se combina con una instrucción **if...else**. Pero por lo general puede obtenerse el **mismo efecto** utilizando los bucles anteriormente vistos.

# Uso de exit

La función `exit()` obliga a la terminación de un programa.

Puede tener como argumento a 0, `exit(0)`, que significa terminación normal. En caso de que tenga como argumento a un número diferente de 0, se supone que puede acceder a analizar el error con ese argumento.

```
#include<stdio.h>
int main(){
    int i;
    printf("\nNumeros del 1 al 100");
    for(i=1;i<101;i++){
        if((i%3==0)&&(i%5==0)) printf("\nFizzBuzz");
        else if(i%3==0) printf("\nFizz");
        else if(i%5==0) printf("\nBuzz");
        else printf("\n%d",i);

        if(i==90) exit(1);
    }
    return 0;
}
```

En este ejemplo obligamos la terminación del programa cuando i sea igual a 90

# Preguntas



# Bibliografía

- [1] “THE C PROGRAMING LANGUAGE” Kermighan y Ritchie – Prentice – Hall, Segunda Edición;
- [2] “FUNDAMENTOS DE PROGRAMACIÓN: ALGORITMOS, ESTRUCTURAS DE DATOS Y OBJETOS” Luis Joyanes Aguilar – McGraw – Hill – Cuarta Edición.
- [3] “COMO PROGRAMAR EN C/C++ Y JAVA” Deitel, Deitel – Prentice – Hall – Cuarta Edición;