# UNIVERSIDAD NACIONAL DE LOMAS DE ZAMORA


## FACULTAD DE INGENIERÍA

## INGENIERÍA MECATRÓNICA


# FUNDAMENTOS DE LOS COMPUTADORES DIGITALES

## MÓDULO 4

## ESTRUCTURAS REPETITIVAS

## CURSO 2021

*Con texto de "Caminando junto al Lenguaje C - Martin Goin - Editorial UNRN – 2016"*

## AUTORES

Martín Gonzalez - Ingeniero en Electrónica (UTN)
Mauro Maceda - Ingeniero en Electrónica (UTN)
Lautaro Fontana - Ingeniero en Mecatrónica (UNLZ)
Ezequiel Blanca - Ingeniero en Electrónica (UTN)
Néstor Maceda - Licenciado en Informática Educativa (UNLa)

## Contenido

# Estructura de control de repetición

Hasta el momento hemos visto ejercicios que no presentan repeticiones de sentencias o acciones. Pero son muy comunes estos casos a la hora de programar. Un ejemplo sencillo de repetición podría ser el ingreso de las temperaturas de cada día de un mes para luego hallar el promedio, o la cantidad de venta semanal de un producto en todo el año para hallar la sumatoria, etcétera.

No es lo mismo hallar el promedio de tres números que el de veinte. Veamos justamente cómo podríamos resolver el ejemplo dado con las herramientas que tenemos hasta el momento para tener una mejor idea.

**Ejemplo 1**: ingresar diez números y hallar su promedio. (Podríamos hacerlo de dos maneras).

**Solución 1**

```
int main(){
    int A,B,C,D,E,F,G,H,I,J;
    float P;
    scanf("%d %d %d %d %d %d %d %d %d %d",&A, &B, &C, &D, &E, &F, &G, &H, &I, &J);
     P = (A+B+C+D+E+F+G+H+I+J)/10;
     printf("El promedio es %f",P);
     return 0;
}
```

**Solución 2**

```
int main() {
    float S = 0;
    int A;
    float P;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
    S = S + A;
    scanf("%d",%A);
 S = S + A;
    scanf("%d",%A);
    S = S + A;
    P = S/10;
    printf("El promedio es %f",P);
    return 0;
}
```

La solución 1 resuelve el problema en pocas líneas pero utiliza una gran cantidad de variables, es decir, mucha memoria, y la solución 2 utiliza pocas variables pero muchas líneas de comando.

Basta imaginar si el algoritmo en vez de resolver el promedio con diez números debiera resolverlo para cien, necesitaríamos para la solución 1 cien variables y para la solución 2, doscientas seis líneas de trabajo.

Para poder revertir dicho problema existen en los lenguajes de programación mecanismos de repetición llamados bucles (*loop*).

Existen tres estructuras o sentencias de control para especificar la repetición: For, While, y Do-While.

## Estructura de control de repetición (sentencia «for»)

Cuando se desea ejecutar un bucle en un determinado número de veces, usamos la sentencia «for».

En estos casos se requiere que conozcamos por anticipado el número de repeticiones.

Podríamos solucionar el problema del ejemplo 1 al saber que un par de acciones se repite diez veces.

```
int main()
{
    float S = 0; //usamos float para prevenir el redondeo en el promedio
    int i;
    float P;
    int A;
    for (i=0; i<10; i++)
    {
        scanf("%d",&A);
        S = S + A;
    }
    P = S/10;
    printf ("El promedio es %f ",P);
    return 0;
}
```

Al saber que las dos sentencias `scanf("%d",&A);` y `S = S + A;` se repiten diez veces, el bucle «for» envuelve dichas acciones.

La sentencia *for* tiene tres parámetros separados por *;* (punto y coma): el primero *i=0* indica el comienzo con valor inicial cero, el segundo indica la condición que debe cumplir, en estos casos debe ser siempre inferior a diez, y el tercero indica el incremento que debe sufrir la variable *i*, en estos casos *i*++ que significa *i = i + 1*, es decir que por cada iteración del bucle se incrementa *i* en uno.

En la primera iteración *i = 0*, luego en la segunda iteración *i = 1*, tercera iteración *i = 2*,.., en la décima *i = 9* y para la decimoprimera vuelta *i = 10*, es decir no cumple con la condición impuesta por el parámetro dos *i<10* de la sentencia *for*, entonces el bucle se termina en ese momento.

Pudo haberse utilizado *para (i=1; i<=10; i++)* siendo equivalente. La variable *i* comienza en uno pero la iteración termina cuando el valor de *i* llega a once. A dicha variable *i* se la reconoce

como *variable de iteración*. No necesariamente debe llamarse así, podría etiquetarse de cualquier otra manera.

También es importante saber que los resultados finales deben ubicarse fuera del ciclo de repetición, por el contrario, estaríamos mostrando en el ejemplo diez veces "El promedio es "…

**Ejemplo 2:** hallar el promedio de altura de los alumnos del curso A y el promedio de los alumnos del curso B y mostrar sus resultados. El curso A tiene 26 alumnos mientras que el B tiene 30.

```
int main()
{
    float SA = 0;
    float SB = 0;
    int altura;
    float PA,PB;
    for (int i=0; i<26; i++)//a partir del estandar C99 se puede declarar I acá
    {
        scanf ("%d",&altura);
        SA = SA + altura;
    }
    for (int i=0; i<30; i++)
    {
        scanf ("%d",&altura);
        SB = SB + altura;
    }
    PA = SA/26;
    PB = SB/30;
    printf("El promedio de altura del curso A es %f",PA);
    printf("El promedio de altura del curso B es %f",PB);
}
```

**Nota:** en estos casos se utilizan dos sentencias *for* separadas, por tener distintas cantidades de iteraciones (26 y 30). Puede utilizarse la misma variable de iteración *i* para los dos for, ya que su contenido vuelve a iniciarse con cero. Lo mismo sucede con la variable *altura* que se repite en ambos ingresos, aunque también pudo haberse utilizado diferentes variables.

Las variables *SA* y *SB* del ejemplo 21 son llamadas **variables sumadoras o acumuladoras**. A continuación explicaremos este tipo de variables especiales.

## Variables contadoras y sumadoras

### Variable contadora

Se trata de una variable que se encuentra en ambos miembros de una asignación a la que se le suma un valor constante (por lo general uno). Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante, cada vez que se produce un determinado suceso, acción o iteración. Los contadores se utilizan con la finalidad de contar

sucesos, acciones o iteraciones internas en un bucle, proceso, subrutina o donde se requiera cuantificar; deben necesariamente ser inicializados antes del comienzo de un ciclo de repetición.
La inicialización consiste en asignarle al contador un valor inicial, es decir, el número desde el cual necesitamos que se inicie el conteo (por lo general comienzan en cero).

**Variable sumadora**

También llamada *acumuladora*, es una variable que se encuentra en ambos miembros de una asignación a la que se le suma un valor variable. Se trata de una variable que, como su nombre lo indica, suma sobre sí misma un conjunto de valores, al finalizar con el ciclo contendrá, en una sola variable, la sumatoria de todos los valores que cumplen una determinada condición (también puede servir para decrementar valores variables). Al igual que el contador es necesario haber inicializado antes del comienzo de un ciclo de repetición.
La inicialización consiste en asignarle al sumador un valor inicial, es decir el número desde el cual necesitamos que se inicie la sumatoria (por lo general comienzan en cero).

**Nota**: la diferencia entre un contador y un sumador es que mientras el primero aumenta en una cantidad fija preestablecida, el acumulador aumenta en una cantidad o valor variable. Y la similitud está en que siempre deben tener un valor inicial antes del bucle de repetición. Seguramente cuando tengamos que hallar promedios o porcentajes necesitaremos de ambas.

A continuación, unos ejemplos:

**Variable contadora**

```
int main()
{
    int C = 0;
    for (int i=0; i < 100; i++)
    {
        if (i % 2 ==0)
            C++; // igual a C=C+1
    }
    printf("La cantidad de n° pares son
%d",C);
    return 0;
}
```

**Variable sumadora o acumuladora**

```
int main()
{
    float S = 0;
    float P;
    for (i=0; i < 100; i++)
    {
        scanf("%d",&A);
        S = S + A; // igual a S+=A
    }
    P = S/100;
    printf("El promedio de altura es
%f",P);
    return 0;
}
```

**Ejemplo 3:** en la salida de un parque nacional se desea saber el promedio de edad de los primeros 100 visitantes que egresan de éste, pero separando los turistas extranjeros de los turistas nacionales.

```
int main()
{
    float SE = 0;
    int CE = 0;
    float SN = 0;
    int CN = 0;
    int altura; //la altura la guardamos en cm de forma entera
    char argentino;
    float PN;
    float PE;
    for (int i=0; i<100; i++)
    {
        scanf ("%d", &altura);
        scanf("%c", &argentino);
        if (argentino == 'S')
        {
            SN = SN + altura; //SN+=altura
            CN++;
        }
        else
        {
            SE = SE + altura;
            CE = CE + 1; //igual a CE++
        }
    }
    PN = SN/CN;
    PE = SE/CE;
    printf ("El promedio de altura de los turistas argentinos es %f",PN);
    printf ("El promedio de edad de los turistas extranjeros es %f",PE);

    return 0;
}
```

**Nota:** en este caso dentro del bucle *for* se encuentra la decisión doble *if* para determinar si la edad ingresada es de un turista nacional o extranjero. Entonces se separan los caminos, si es verdadera la condición *(argentino == 'S')* entonces toma el camino del *bloque por verdadero (if)*, caso contrario, el camino del *else*.

Al no saber de entrada la cantidad de turistas nacionales y extranjeros, fue necesario usar *CE* y *CN* llamadas *variables contadoras*, ya que cada vez que pasan por *CE = CE + 1* y *CN = CN + 1* sus valores se incrementan en uno. Éstas, al igual que las sumadoras, deben inicializarse en cero antes del ciclo de repetición.

**Aclaración:** en la asignación de variables contadoras puede escribirse de un modo reducido.

Por ejemplo *CN = CN + 1* es equivalente a *CN++*

A partir de ahora lo utilizaremos del modo reducido.

Es posible decrementar el valor de la variable de iteración.

**Ejemplo 4:** mostrar de modo decreciente los números del N al 1. El usuario debe ingresar dicho número.

```
int main()
{
    int N;
    scanf("%d",&N);
    for (int i=N; i>0; i--)
    {
        printf("%d ",i);
    }
    return 0;
}
```

## Bucle for anidado

Al igual que sucede con la sentencias de decisión if, los bucles pueden estar anidados, es decir uno dentro de otro. Se debe tener cuidado con que el bucle interior esté completamente dentro del bucle exterior. Si se cruzan, se produce un error. En pocas palabras, hay que colocar correctamente las llaves que abren y cierran.

A continuación se muestra en un cuadro un ejemplo genérico que compara lo incorrecto con lo correcto para los cierres de llaves *{}* en bucles *for.*

| Incorrecto | Correcto |
|---|---|
| `for (int i=0; i<N; i++)`<br>`{`<br>`for (int j=0; j<M; j++)`<br>`}`<br>`{`<br>`}` | `for (int i=0; i<N; i++)`<br>`{`<br>`for (int j=0; j<M; j++)`<br>`{`<br>`}`<br>`}` |

**Ejemplo 5:** realizar un algoritmo que produzca por pantalla lo siguiente:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 2 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 3 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 4 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 5 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 6 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 7 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 8 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 9 | → | 100 | 110 | 120 | 130 | 140 | 150 |
| 10 | → | 100 | 110 | 120 | 130 | 140 | 150 |

```
int main()
{
    for (int i=1; i <= 10; i++)
    {
        printf ("%d ----> ",i);
        for (int j=100; j<=150; j=j+10)
        {
            printf("%d     ",j);
        }
        printf("\n"); //imprime un salto de linea
    }
    return 0;
}
```

**Nota:** el primer bucle *for* se incrementa en uno y el segundo, de diez en diez *j = j + 10*, comenzando con *j=100*, mientras sea *j <= 150*. En este caso sí es importante tener distintas variables de iteración *(i , j)*.

También es importante ver cómo fueron cerrados los bucles por sus llaves. Observemos que la primera sentencia *printf ("%d ----> ",i);* se realiza para la primera columna de la pantalla, el número de iteración y una flecha, luego dentro del segundo bucle *for* se imprimen los cinco números del 100 al 150. El último *printf("\n");* que está fuera del segundo pero dentro del primero, sirve para hacer el salto de línea, como si fuera un enter. Si no fuera por esta última instrucción, la pantalla tendrá el siguiente aspecto: 1 → 100 110 120 130 140 150 2 → 100 120 130 140 150 3 → 100 … 10 → 100 110 120 130 140 150 . Es decir, todo en una única línea.

El ejemplo anterior muestra cómo se utilizan dos *for* anidados, su uso es frecuente cuando se trabaja con matrices.

Es posible anidar más de dos bucles *for.*

## Máximos y mínimos

Para hallar el valor máximo y/o mínimo de una lista de valores numéricos es necesario utilizar una variable adicional. Mostramos un ejemplo a continuación.

**Ejemplo 6:** hallar la temperatura máxima registrada en una laguna sabiendo que se toman 30 muestras.

```
int main()
{
    float Tmax = -10000;
    float T;
    for (int i=0; i < 30; i++)
    {
        scanf("%f",&T);
        if (T > Tmax)
            Tmax = T;
    }
    printf("La mayor temperatura registrada es %f",Tmax);
    return 0;
}
```

**Nota:** el valor inicial de la variable *Tmax* es un número extremadamente pequeño (un absurdo para la temperatura de una laguna), cuando se ingresa el primer valor de temperatura dentro del bucle, la condición *if* se efectúa al ser verdadera, entonces el valor de *Tmax* cambia por el valor de la temperatura *T*.

A medida que se ingresan las otras temperaturas, la sentencia *si* evalúa si aparece un valor mayor o no a *Tmax*, en caso afirmativo, se le asigna a éste el nuevo valor de *T*.

Tomar en cuenta que si se produce el caso de empate la condición del *if* es falsa, por lo tanto, no sufre efecto.

Si por lo contrario, el ejercicio tratara de hallar la mínima temperatura de la laguna, entonces debemos cambiar *Tmax* por *Tmin*, asignarle un valor extremadamente grande (un absurdo, por ejemplo *Tmin = 10000*) y la condición de la decisión *if* será *si (T<Tmin)*. A continuación, mostramos un ejemplo de cómo se halla ambas.

```
int main()
{
    float Tmax = -10000;
    float Tmin = 10000;
    float T;
    for (int i=0; i < 30; i++)
    {
        scanf("%f",&T);
        if (T > Tmax)
            Tmax = T;
        if (T < Tmin)
            Tmin = T;
    }
    printf("La mayor temperatura registrada es %f",Tmax);
    printf("La menor temperatura registrada es %f",Tmin);
    return 0;
}
```

**Nota:** observar que las sentencias ***if*** trabajan de modo independiente al no ser excluyentes. Podría pasar que tanto el máximo como el mínimo coincidan, esto se produce cuando las treinta temperaturas son exactamente iguales.

¿Qué sucede si además de registrar la máxima temperatura de la laguna, quisiéramos saber cuál de todas las muestras se registró?

**Ejemplo 7:** hallar la temperatura máxima registrada en una laguna sabiendo que se toman 30 muestras y en qué muestra fue registrada.

```
int main()
{
    float Tmax = -10000;
    float T;
    int Tmuestra;
    for (int i=0; i < 30; i++)
    {
        scanf("%f",&T);
        if (T > Tmax)
        {
            Tmax = T;
            Tmuestra = i+1;
        }
    }
    printf("La mayor temperatura registrada es  % f y se registró en la muestra n°
%f",Tmax, Tmuestra);
    return 0;
}
```

**Nota:** a la variable *Tmuestra* se le asigna el valor de la variable de iteración *i* del *for* más uno, ya que empieza del cero. Por ejemplo si la temperatura máxima se produce en la vuelta 8 significa que i tiene valor 7 en dicha vuelta, entonces es necesario incrementarlo.

De esta manera, se tiene el registro de cuando se produce la última condición verdadera de la sentencia *if*.

En caso de empate el único registro que se produce es el primero que llega.

## Estructura de control de repetición (sentencia «while»)

Esta estructura repite una instrucción o grupo de instrucciones mientras una expresión lógica sea cierta.
Cuando no se conoce el número de repeticiones por anticipado, la sentencia *while* lo resuelve con el empleo de una determinada condición.
Lo primero que hace esta sentencia es evaluar si se ejecuta el bucle, es decir que es posible que nunca se realice acción alguna.

Mientras la condición se cumpla, el bucle sigue iterando, por eso es importante no caer en ciclos de repetición infinitos.

**Ejemplo 8:** calcular la suma de números ingresados por teclado hasta que se ingrese un cero.

```
int main()
{
    int S = 0;
    int N;
    scanf("%d",&N);
    while ( N != 0)
    {
        S = S + N;
        scanf("%d",&N);
    }
    printf("La sumatoria es  %d",S);
    return 0;
}
```

**Nota:** es necesario ingresar *N* antes de arrancar con el *while,* porque debe evaluarse la condición al entrar al bucle *while,* luego es importante no olvidarse de ingresar *N* dentro del bucle, porque si no lo hacemos caeremos en un ciclo de repetición infinito.

¿Cuál será el resultado si arrancamos con el ingreso de un cero?

Dentro del *while* pueden utilizarse condiciones compuestas como veremos a continuación.

**Ejemplo 9:** hallar el promedio de números ingresados por teclado hasta que aparezca uno que no sea par y mayor a 20.

```
int main()
{
    float S = 0;
    int C = 0;
    int N;
    float P;
    scanf("%d",&N);
    while ( !( (N % 2 == 1) && (N > 20) ) )
    {
        S = S + N;
        C++;
        scanf("%d",&N);
    }
    P = S/C;
    printf("El promedio es %f",P);
    return 0;
}
```

**Nota:** como no sabemos de entrada la cantidad de números que se van a ingresar, entonces debemos usar un contador *C* para poder finalmente hallar el promedio. El inconveniente que puede presentar este algoritmo es que de entrada no se cumpla la condición del *while*, por lo tanto el valor de *C* terminaría en cero y generaría un error en la división al hallar el promedio *P*. Esto se podría resolver agregando antes de dicho cálculo la siguiente sentencia *if*.

```
int main()
{
    float S = 0;
    int C = 0;
    int N;
    float P;
    scanf("%d",&N);
    while ( !( (N % 2 == 1) && (N > 20) ) )
    {
        S = S + N;
        C++;
        scanf("%d",&N);
    }
    if(C == 0)
        printf("No hubo números de ingreso");
    else
    {
        P = S/C;
        printf ("El promedio es %f ",P);
    }

    return 0;
}
```

**Ejemplo 10:** contar la cantidad de veces que se ingresan números pares y la cantidad de números impares hasta que se ingrese un número negativo. El cero no se cuenta.

```
int main()
{
    int CP = 0;
    int CI = 0;
    int N;
    scanf("%d",&N);
    while ( N > = 0 )
    {
        if ((N % 2 == 0) && (N != 0))
            CP++;
        else if ((N % 2 == 1) && (N != 0))
            CI++;
        scanf("%d",&N);
    }
    printf("La cantidad de números pares es %d",CP);
    printf("La cantidad de números impares es %d",CI);
    return 0;
}
```

**Nota:** mientras *N* no sea negativo el bucle sigue iterando. Para incrementar los contadores se les agregó a las condiciones *(N != 0)* porque el enunciado del problema así lo determinaba.

Al igual que la sentencia *for* y la sentencia *if*, pueden anidarse los *while*

## Estructura de control de repetición (sentencia «do while»)

Esta estructura repite una instrucción o grupo de instrucciones mientras una expresión lógica sea cierta.

Un aspecto muy importante de la presente estructura de control es que la expresión lógica no se evalúa hasta el final de la estructura con lo cual el bucle se ejecuta al menos una vez, contraria a la estructura «while» que podía no ejecutarse nunca. Es decir que después de cada iteración el bucle evalúa la condición, si es verdadera sigue repitiendo y si la condición es falsa termina.

**Ejemplo 11:** hacer un algoritmo que muestre del 0 al 20 (solo los números pares).

```
int main()
{
    int N = 0;
    do
    {
        printf("%d \n",N);
        N = N + 2;
    }
    while ( N < 21 );
}
```

**Nota:** a medida que itera, el bucle se incrementa *N* de dos en dos, en la última repetición se imprime 20 y se incrementa en 22, por lo tanto, termina porque la condición ahora es falsa.

**Ejemplo 12:** en un bosque se necesita saber el promedio de diámetro de cada tronco de ciprés y el promedio de su altura. El proceso termina cuando el usuario responde con una 'N', mientras tanto, debe responder con 'S'.

```
int main()
{
    int A,D;
    char opcion;
    int C = 0;
    float SA = 0;
    float SD = 0;
    do
    {
        printf("Ingrese su altura: ");
        scanf("%d",&A);
        printf("Ingrese su diametro: ");
```

```
        scanf("%d",&D);
        SA = SA + A;
        SD = SD + D;
        C++;
        printf("¿Desea seguir ingresando datos? S/N \n");
        scanf(" %c",&opcion);
    }
    while(opcion=='S');
    float PA = SA/C;
    float PD = SD/C;
    printf ("El promedio de altura de los cipreses es %f \n",PA);
    printf ("El promedio de diámetro de los cipreses es %f \n",PD);
    return 0;
}
```

**Nota:** la variable *opción* está reservada para que el usuario responda S o N (si quiere seguir ingresando o no).

Tomar en cuenta que para seguir ingresando datos el usuario debe presionar solo la S (mayúscula). Para resolver este problema podría ponerse la siguiente condición en el *mientras*

*(opción == 'S' && opción == 's')*

Al igual que en las anteriores estructuras, es posible crear algoritmos que tengan bucles «do while» una dentro de otra.

# Tabla de estructura de control de repetición

| Sintaxis | Significado | Ejemplo |
|---|---|---|
| `for (valor inicial; condición;incremento/ decremento)`<br>`{`<br>`sentencias;`<br>`}` | Bucle de repetición exacto.<br>Se sabe de ante- mano la cantidad de iteraciones que cumple el bucle. | `for(int i=1; i <= 10; i++)`<br>`{`<br>`printf("3 X %d = %d \n",i,i*3);`<br>`}`<br>Imprime la tabla de multiplicación del 3. |
| `while(condición)`<br>`{`<br>`sentencias;`<br>`}` | Bucle de repetición condicional. Mientras se cumpla la condición se repiten las sentencias dentro del bucle. Puede ocurrir que no exista ejecución de sentencias de entrada. | `int i = 1;`<br>`while (i<=10)`<br>`{`<br>`printf("3 X %d = %d \n",i,i*3);`<br>`i++;`<br>`}`<br>Imprime la tabla de mul- tiplicación del 3. |
| `do`<br>`{`<br>`sentencias;`<br>`}`<br>`while(condición);` | Bucle de repetición condicional.<br>Mientras se cumpla la condición se repiten las sentencias dentro del bucle. Pero siempre se ejecuta una vez como mínimo, porque la condición se encuentra al finalizar las sentencias. | `int i = 1;`<br>`do`<br>`{`<br>`printf("3 X %d = %d \n",i,i*3);`<br>`i++;`<br>`}`<br>`while (i<=10);`<br>Imprime la tabla de multiplicación del 3. |