



PROGRAMACIÓN ESTRUCTURADA

Unidad 2.- Estructuras de Control de Flujo



COMPETENCIA:

Desarrolla programas computacionales, utilizando las estructuras de control de flujo de la programación estructurada.

SECUENCIA DE CONTENIDOS:

1. Funciones de Entrada/ Salida.
2. Estructuras selectivas de control de flujo de un programa.
3. Estructuras repetitivas de control de flujo de un programa.



1.- Funciones de Entrada/Salida



Funciones de Entrada / Salida

- Todo programa en C interactúa a través de datos de entrada y salida.
- En C se utiliza la librería *stdio.h* para el manejo de E/S.
- En C la entrada y salida se lee y escribe de los dispositivos estándar de entrada y salida denominados: `stdin` y `stdout` respectivamente.



Funciones de Entrada / Salida

- La salida normalmente es a pantalla y la entrada es por medio del teclado.
 - Los datos pueden ser lanzados a impresoras, discos, etc.
- `stdio.h` define macros, constantes, variables, y funciones que permiten intercambiar datos.
- Existe un flujo estándar de error: *stderr*, se usa para desplegar mensajes de error.



Función *getchar()*

- Se utiliza para leer carácter por carácter.
- Devuelve el carácter siguiente del flujo de entrada *stdin*.
- En caso de error, o de encontrar el fin de archivo, devuelve EOF.

- Ejemplo:

```
#include <stdio.h>
int main(){
    int car;
    int cuenta=0;
    while ((car=getchar()) !='\n')
        if (car == 'a') ++cuenta;
    printf("%d letras a \n", cuenta);
    return 0;
}
```



Función *putchar()*

- Es la función opuesta a `getchar()`.
- Se utiliza para escribir en la salida (`stdout`) carácter a carácter.

■ Ejemplo:

```
#include <stdio.h>  
#include <ctype.h>  
  
int main(){  
    char car;  
    car =getchar();  
    if (car == 'a')  
        putchar(toupper(car));  
    else  
        putchar(car);  
    return 0;  
}
```



Función *gets()* y *puts()*

■ Función *gets()*

- Obtiene una cadena de caracteres del teclado.
- Lee caracteres, incluyendo blancos de separación entre palabras, hasta encontrar el carácter de fin de línea o fin de archivo.
- El argumento de *gets()* es una variable cadena de suficiente espacio para que puedan guardarse todos los caracteres tecleados.
- El carácter de fin de línea no se incluye en la variable cadena.

■ Función *puts()*

- Es la función opuesta de *gets()*.
- Visualiza una cadena de caracteres, incluyendo el carácter de fin de línea.



Ejemplo:

Solicita introducir un nombre, comprueba la operación y lo escribe en pantalla.

```
#include <stdio.h>  
#include <conio.h>  
int main(){  
    char nombre[80];  
    puts("\nIntroduce tu nombre: ");  
    gets(nombre);  
    puts("\nHola ¿Como estas? ");  
    puts(nombre);  
    return 0;  
}
```



Función *printf()*

- La función `printf()` visualiza en la pantalla datos del programa, transforma los datos, que están en representación binaria, a ASCII según los códigos transmitidos. Su sintaxis es la siguiente:

`printf(cadena de control, dato1, dato2, ...);`

- Ejemplo:

`a=9, b=8, c='a', d= 12.80049`

`printf("a=%d b=%d c=%c d=%f", a, b, c, d);`
`printf("%.3f", d);`

- El número de argumentos de `printf()` es indefinido, por lo que se pueden transmitir cuantos datos se deseen.



Funciones *scanf*

- La entrada de datos a un programa puede tener diversas fuentes, teclado, archivos, disco, etc. La función más utilizada, por su versatilidad, para entrada formateada es `scanf()`. Su sintaxis es la siguiente:

`scanf(cad_control, var1, var2, var3...);`

- Ejemplo:

```
int n; double x;  
scanf("%d %lf", &n, &x); //La entrada tiene que ser de la forma: 134 -1.4E-4
```
- Los argumentos `var1`, `var2`, ... de la función `scanf()` se pasan por dirección o referencia pues van a ser modificados por la función para devolver los datos. Por ello necesitan el operador dirección, el prefijo `&`.



Códigos de formato

- `%d` el dato se convierte a entero decimal.
- `%o` el dato se convierte a entero octal.
- `%x` el dato se convierte a entero hexadecimal.
- `%u` el dato se convierte a entero sin signo.
- `%c` el dato se considera tipo carácter.
- `%e` el dato se considera de tipo float. Se convierte a notación científica de la forma `{-}n.mmmmmmmE{+/-}dd`.
- `%f` el dato se considera de tipo float. Se convierte a notación decimal, con parte entera y los dígitos de precisión.
- `%lf` el dato se considera de tipo double.
- `%g` el dato se considera de tipo float. Se convierte al código `%e` o `%f` dependiendo de la representación más corta.
- `%s` el dato se considera una cadena de caracteres.



Códigos de formato

TIPO DE ARGUMENTO	CARÁCTER DE FORMATO	FORMATO DE SALIDA
<i>Númérico</i>	<i>%d</i>	<i>signed decimal int</i>
	<i>%i</i>	<i>signed decimal int</i>
	<i>%o</i>	<i>unsigned octal int</i>
	<i>%u</i>	<i>unsigned decimal int</i>
	<i>%x</i>	<i>unsigned hexadecimal int (con a, ..., f)</i>
	<i>%X</i>	<i>unsigned hexadecimal int (con A, ..., F)</i>
	<i>%f</i>	<i>[-]dddd.dddd</i>
	<i>%e</i>	<i>[-]d.dddd o bien e[+/-]ddd</i>
	<i>%g</i>	<i>el más corto de %e y %f</i>
	<i>%E</i>	<i>[-]d.dddd o bien E[+/-]ddd</i>
	<i>%G</i>	<i>el más corto de %E y %f</i>
<i>Carácter</i>	<i>%c</i>	<i>carácter simple</i>
	<i>%s</i>	<i>cadena de caracteres</i>
	<i>%%</i>	<i>el carácter %</i>
<i>Punteros</i>	<i>%n</i>	<i>se refieren a punteros y se</i>
	<i>%p</i>	<i>estudiarán en el Capítulo 14</i>



Secuencias de escape

- C utiliza secuencias de escape para visualizar caracteres que no están representados por símbolos tradicionales. Las secuencias de escape proporcionan flexibilidad en las aplicaciones mediante efectos especiales. Las secuencias de escape clásicas se muestran a continuación:
 - `\a` alarma
 - `\b` Retroceso de espacio
 - `\f` Avance de página
 - `\n` Retorno de carro y avance de línea
 - `\r` Retorno de carro
 - `\t` Tabulación
 - `\v` Tabulación vertical
 - `\\` barra inclinada
 - `\?` Signos de interrogación
 - `\"` Dobles comillas
 - `\000` Número octal
 - `\xhh` número hexadecimal



Uso de fflush

- Cuando se lee una cadena y luego un valor numérico, el "Enter" que se pulso para la cadena, se queda en el buffer de entrada y enviado a la siguiente lectura.
- Se requiere vaciar el buffer antes de otra lectura, con fflush.

```
#include <stdio.h>
```

```
int main() {  
    char nombre[80];  
    int edad;  
    printf("Nombre: ");  
    scanf("%s",nombre);  
    fflush(stdin);  
    printf("Edad: ");  
    scanf("%d",&edad);  
    printf ("\n%d",edad);  
    return 0;  
}
```



Función *sscanf()*

- La función `scanf()` (*scan-format*, analizar con formato), en realidad representa a una familia de funciones que analizan una entrada de datos con formato y cargan el resultado en los argumentos que se pasan por referencia a dicha función o funciones:
 - La función `scanf()` lee los datos de entrada en el **stdin** (flujo de entrada estándar).
 - La función `fscanf()` (*file-scanf*) lee en un flujo de entrada dado, por lo general un fichero (file) abierto para lectura.
 - La función `sscanf()` (*string-scanf*) obtiene la entrada que se va a analizar de una cadena de caracteres dada (*string*).



Función *sscanf()*

- Todas ellas leen caracteres, los interpretan según un formato, y almacenan los resultados en sus argumentos. Cada uno cuenta con varios argumentos: por un lado, un formato de la secuencia del control (se describe más abajo), por otro, un sistema de argumentos del indicador que señala dónde la entrada convertida debe ser almacenada. El resultado es indefinido si hay escasos argumentos para dar formato. Si se agota el formato mientras que sigue habiendo los argumentos, los argumentos sobrantes son evaluados pero no procesados de ninguna otra manera.



Función *sprintf()*

- La función `sprintf()` es similar a `printf()`, la diferencia es que `printf` envía su resultado a un objeto de archivo, mientras que `sprintf` regresa una cadena del resultado del `printf`.
- Por otro lado, esta función es equivalente a `fprintf`, excepto que el argumento **cadena** especifica un array en el cual la salida generada es para ser escrita, en vez de un stream. Un carácter nulo es escrito al final de los caracteres escritos; no es contado como parte de la suma retornada. El comportamiento acerca de copiando entre objetos que se superponen no está definido.

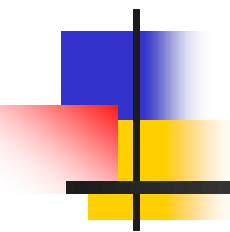


Función *sprintf()*

■ Ejemplo:

```
#include <stdio.h>
int main() \{
    char nombre[20], mensaje[81];
    unsigned int edad=0;
    printf( "Escriba su nombre: " );
    scanf( "%s", nombre );
    printf( "Escriba su edad: " );
    scanf( "%u", &edad );

    sprintf( mensaje, "\nHola %s. Tienes %d anyos.\n", nombre, edad );
    puts( mensaje );
    return 0;
}
```



2.- Estructuras selectivas de control de flujo de un programa.

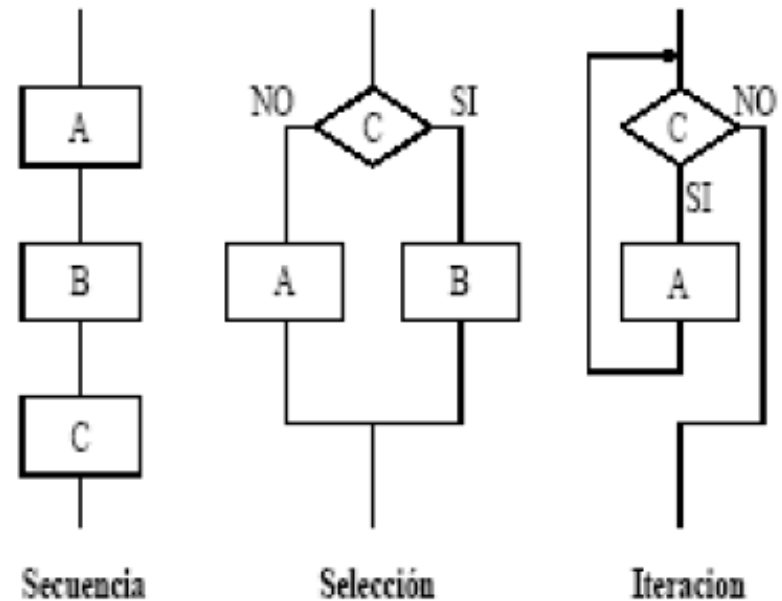


Falso y verdadero en C

- C no tiene tipos de datos lógicos o booleanos para representar los valores verdadero (true) y falso (false). En su lugar se utiliza el tipo int para este propósito, con el valor entero 0 que representa a falso y distinto de cero a verdadero.
 - Not (!): produce falso si su operador es verdadero.
 - And (& &): produce verdadero sólo si ambos operandos son verdaderos, si cualquiera de los operandos es falso, produce falso.
 - Or (||): produce verdadero si cualquiera de los operandos es verdadero y produce falso sólo si ambos operandos son falsos.

Estructuras de Control

- Controlan el flujo de ejecución de un programa o función.
- Permiten combinar instrucciones o sentencias individuales en una simple unidad lógica con un punto de entrada y un punto de salida.
- Existen tres tipos: secuencia, selección y repetición.





Sentencia compuesta

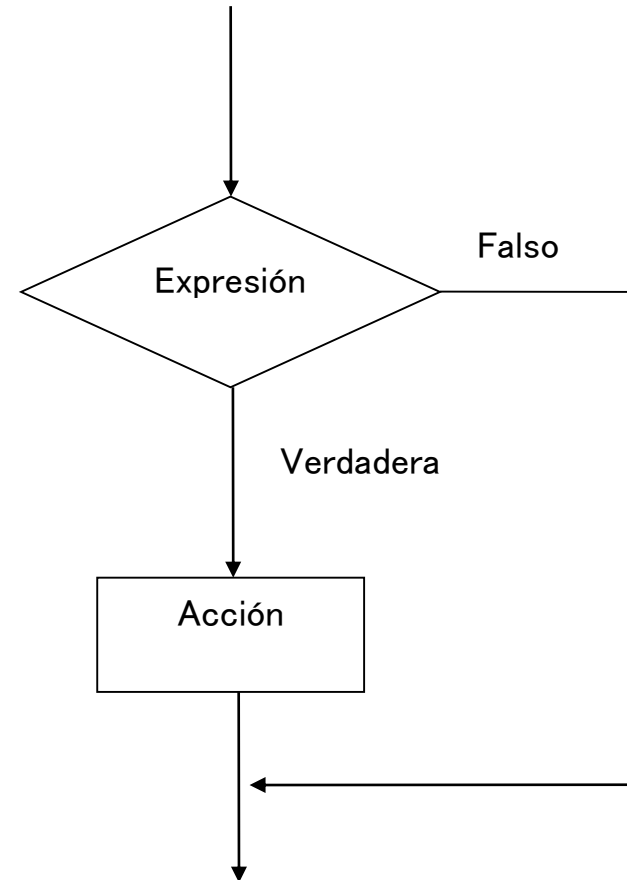
- Conjunto de sentencias encerradas entre llaves ({ y }).
- Las { } se utilizan para especificar un bloque de sentencias con un flujo secuencial.

```
{  
    sentencia1;  
    sentencia2;  
    ...  
    sentenciaN;  
}
```

- El control fluye de la sentencia1 a la sentenciaN.

La sentencia *if*

- La sentencia *if* es una estructura de control de selección, es decir, hay una bifurcación en las opciones.
- Al llegar a la sentencia *if* en un programa, se evalúa una expresión entre paréntesis que viene a continuación del *if*. Si la expresión es verdadera (distinto de cero), se ejecuta acción; en caso contrario (cero) no se ejecuta acción.





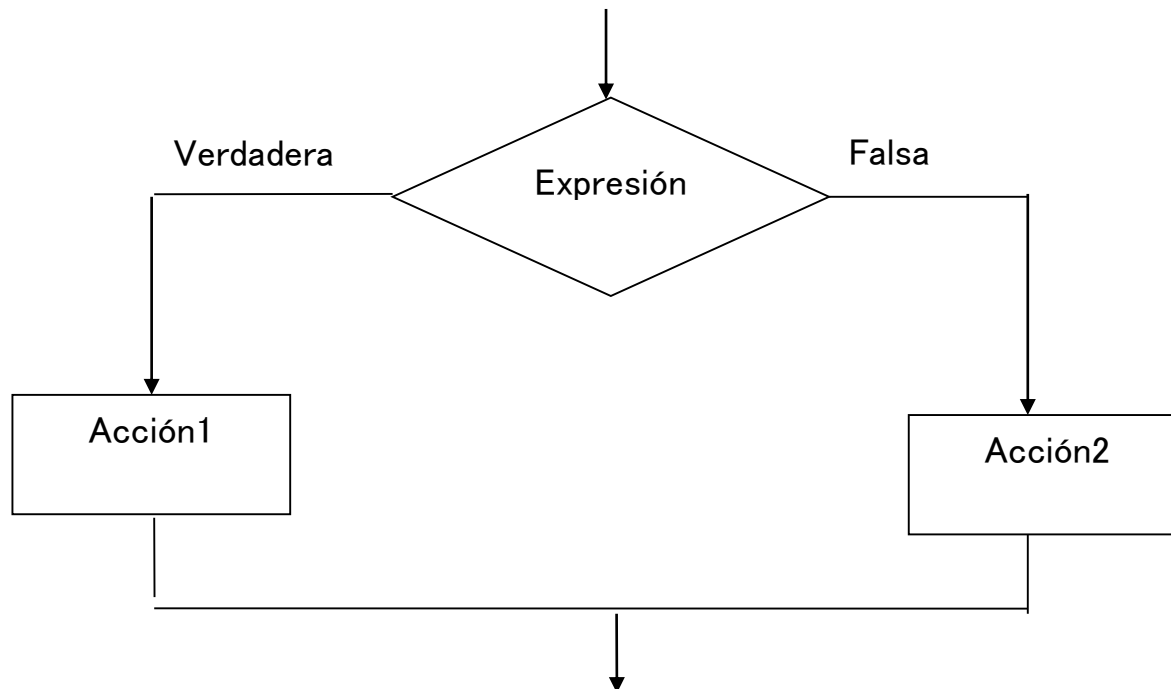
Ejemplo

- Pedir dos enteros al usuario y verificar si el primer entero es divisible por el segundo:

```
int main(){  
    int a,b;  
    printf("Introduzca dos enteros");  
    scanf("%d %d", &a,&b);  
    if(a%b==0)  
        printf(" %d es divisible por %d\n", a, b);  
    return 0;  
}
```

La sentencia *if / else*

- Otro formato de la sentencia *if* es *if-else*.
- Al llegar a la sentencia *if* en un programa, se evalúa una expresión entre paréntesis que viene a continuación del *if*. Si la expresión es verdadera (distinto de cero), se ejecuta acción1; en caso contrario (cero) se ejecuta acción2.





Sentencia *switch*

- La sentencia *switch* se utiliza para seleccionar una de múltiples alternativas.
- Es útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada *expresión de control o selector*.
- El valor de la expresión en la sentencia *switch* puede ser de tipo *int* o *char*, pero no de tipo *float* ni *double*.

```
switch (selector)
{
    case etiqueta1: sentencias1;
        break;
    case etiqueta2: sentencias2;
        break;
    .
    .
    .
    case etiquetan: sentenciasn;
        break;
    default: sentenciasD; //opcional
}
```



Ejemplos

- Elección de tres alternativas y un valor por defecto:

```
switch(opcion)  
{  
  case 0: puts("Cero!");  
    break;  
  case 1: puts("Uno!");  
    break;  
  case 2: puts("Dos!");  
    break;  
  default:  
    puts("Fuera de rango");  
}
```

- Selección múltiple, tres etiquetas ejecutan la misma sentencia:

```
switch(opcion)  
{  
  case 0:  
  case 1:  
  case 2: puts("Menor de 3");  
    break;  
  case 3: puts("Igual a 3");  
    break;  
  default:  
    puts("Mayor que 3");  
}
```

- Dada una nota de un examen mediante un código escribir el literal que le corresponde a la nota.



Expresión condicional: operador ? :

- Es un tercer mecanismo de selección que tiene el siguiente formato:

condición ? expresión1 : expresión2

- Se evalúa la condición, si el valor de la condición es verdadera (distinto de cero) entonces se devuelve como resultado el valor de la expresión1; si el valor de la condición es falsa (cero) se devuelve como resultado el valor de la expresión2.
- Ejemplo:
 - Seleccionar el mayor de dos números enteros:

n1 > n2 ? printf("%d > %d", n1,n2) : printf("%d <= %d", n1,n2);



3.- Estructuras repetitivas de control de flujo de un programa



Sentencia *for*

- La sentencia ***for*** es un método para ejecutar un bloque de sentencias un número fijo de veces.
- Sintaxis:
for (inicialización; condiciónIteración; Incremento)
sentencias;

Donde:

- Inicialización: Inicializa la variable de control del ciclo
- condiciónIteración: Expresión lógica que determina si las sentencias se han de ejecutar (mientras sea verdadera)
- Incremento: Incrementa o decrementa la variable de control del ciclo.
- Sentencias: Sentencias a ejecutar en cada iteración del ciclo.



Ejemplos

- Escribir "Hola!" y el valor de la variable de control.

```
int i;  
for(i=0; i<10; i++){  
    printf("Hola!\n");  
    printf("El valor de i es: %d ", i);  
}
```




Sentencia *for*

- Incremento:

```
int n;  
for(n=1; n<=10; n++){  
    printf("%d | t %d | n ", n, n*n);  
}
```

- Decremento:

```
int n;  
for(n=10; n>5; n--){  
    printf("%d | t %d | n ", n, n*n);  
}
```



Ejemplos

```
int i;
for(i=1; i<100; i*=2){
    printf("%d", i);
}
```

```
int c;
for(c='A'; c<='Z'; c++){
    printf("%c", c);
}
```

```
#define MAX 25
int i, j ;
for(i=0; i=MAX, i<j; i++, j--){
    printf("%d", (i + 2 * j));
}
```

```
double p;
for(p=0.75; p<=5; p+=0.25){
    printf("Perímetro es igual a %.2lf ", p);
}
```

```
int i;
for(i=9; i>=0; i-=3){
    printf("%d", (i * i));
}
```

```
double x;
for(x=pow(y,3.0); x > 2.0; x = sqrt(x)){
    printf("x es ahora igual a %.5e", x);
}
```



Ciclos infinitos

```
for(;;)
    sentencia;

for(;;)
    printf("Hola mundo");

for(;;){
    lista_sentencias1;
    if (condición_terminación) break;
    lista_sentencias2;
}
```

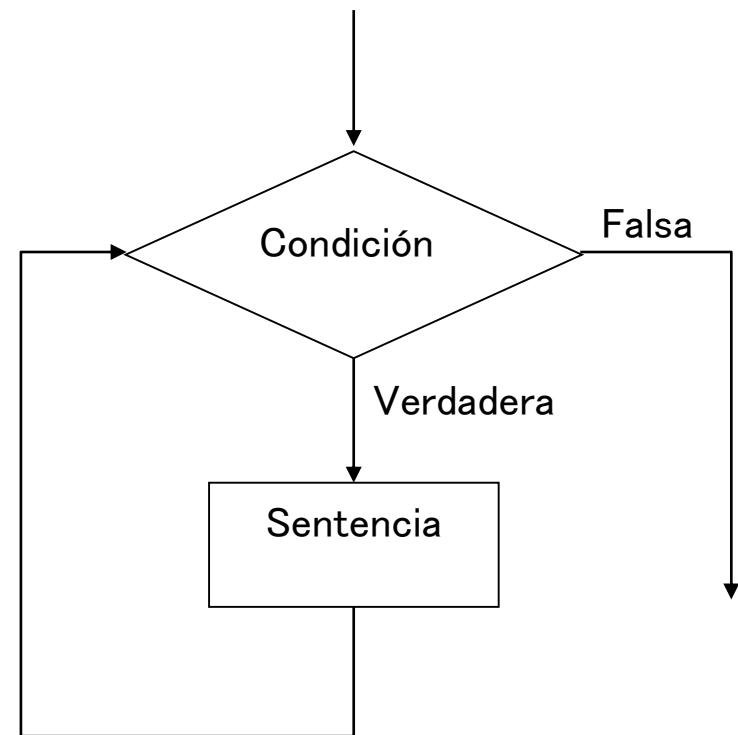
```
#define CLAVE -999
for(;;){
    printf("Introduzca un número, (%d) para terminar", CLAVE);
    scanf("%d", &num);
    if(num == CLAVE) break;
    ...
}
```



v. La sentencia while y do ... while

Sentencia *while*

- Tiene una condición del ciclo (expresión lógica) que controla la sentencia de repetición.
- La posición de la condición es delante del cuerpo del bucle.
- Un ciclo while es un bucle pretest, es decir, la condición se evalúa antes de que se ejecute el cuerpo del ciclo.





Sintaxis *while*

```
while (condicion)  
{  
    sentencia-1;  
    sentencia-2;  
    .  
    .  
    .  
    sentencia-n;  
}
```

Se evalúa la condición, si la condición es verdadera (distinto de cero), la sentencia especificada (cuerpo del ciclo) se ejecuta y se evalúa nuevamente la condición; en caso contrario, el control se transfiere a la sentencia siguiente al ciclo *while*.

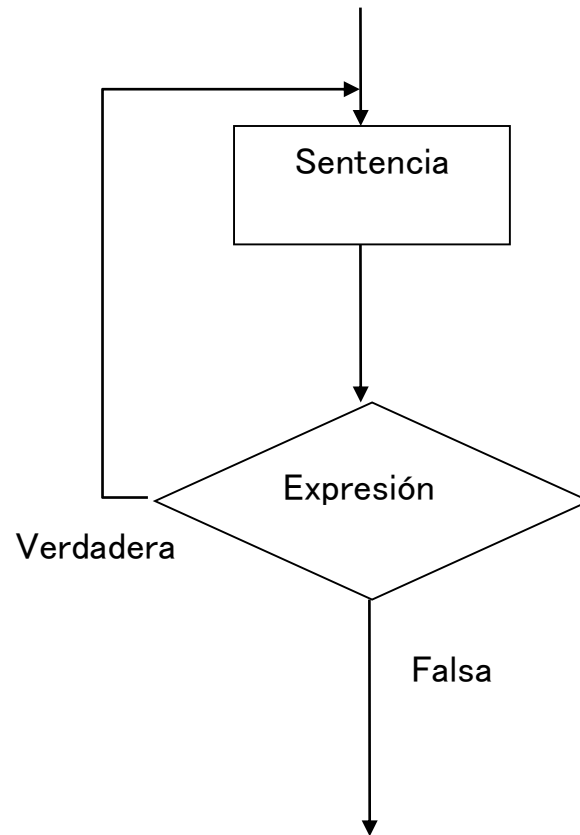


Ejemplo

```
int contador = 0;
while (contador < 5)
{
    contador++;
    printf("contador: %d \n", contador);
}
printf("Terminado, contador = %d\n", contador);
contador*=contador;
printf("contador = %d\n", contador);
```

Sentencia *do...while*

- Se utiliza para especificar un bucle condicional que se ejecuta al menos una vez.
- Después de cada ejecución de sentencia se evalúa expresión:
 - si es falsa, se termina el ciclo y se ejecuta la siguiente sentencia;
 - si es verdadera se repite el cuerpo del ciclo (la sentencia).





Sentencia *do...while*

- Sintaxis:

do
sentencia
while (expresión)

- Ejemplo:

- seleccionar una opción de saludo al usuario dentro de un programa .

```
char opcion;  
do{  
    puts("Hola");  
    puts(" ¿Desea otro tipo de saludo?");  
    puts("Pulse s para si y n para no, ");  
    puts("y a continuacion pulse enter ");  
    scanf(" %c",&opcion);  
}while (opcion == 's' || opcion == 'S');  
puts("Bye");
```



Ejercicios

¿Cuál es el valor de consumo?

```
int consumo, velocidad =120;
if (velocidad > 80)
    consumo=10;
else if (velocidad > 100)
    consumo=12;
else if (velocidad > 120)
    consumo = 15;
printf("%d", consumo);
```

¿Son correctas las siguientes instrucciones?

```
if (x=0) printf("%d = 0 \n",x)
else printf("%d != 0 \n", x);
```

```
if (x < y < z)
    printf("%d <%d < %d\n",x,y,z);
```

¿Qué salida producirá el siguiente código si primera_opcion= 1, 2, 5?

```
int primera_opcion;
switch (primera_opcion +1){
    case 1:
        puts ("Cordero asado");
        break;
    case 2:
        puts ("Chuleta frita");
        break;
    case 3:
        puts ("Chuletón");
        break;
    case 4:
        puts ("Postre de pastes");
        break;
    default:
        puts("Buen apetito");
}
```

¿Qué diferencia hay entre cada bloque?

```
int x=0;
if (x>=0)
    x++;
else if (x>=1);
    x+=2;
```

```
int x=0;
if (x>=0)
    x++;
if (x>=1)
    x+=2;
```

```
int x=0;
if (x>=0)
    x++;
else if (x>=1)
    x+=2;
```



Sentencia *continue*

- *Continue* hace que la ejecución de un ciclo vuelva a la cabecera del ciclo.
- El siguiente ejemplo utiliza *continue* en un ciclo para que si se cumple la condición de la sentencia *if* vuelva a la cabecera e incremente *i* en 1 (*i++*).

```
int clave, i;  
puts("Introduce -9 para acabar ");  
clave = 1;  
for(i=0;i<8;i++){          //ciclo for que no termina  
if(clave==-9) continue;  
scanf("%d",&clave);  
printf(" clave %d\n", clave);  
}  
printf("Valores Finales i=%d clave = %d", i, clave);
```



Sentencia *break*

- La sentencia *break* se utiliza, a veces, para realizar una terminación anormal del bucle. Dicho de otro modo, una terminación antes de lo previsto.
- La sentencia *break* se utiliza para la salida de un bucle *while* o *do-while*, aunque también se puede utilizar dentro de una sentencia *switch*, siendo éste su uso más frecuente.
- Ejemplo:

```
int clave=-9;  
int entrada;  
While (scanf("%d", &entrada)){  
    if (entrada !=clave)  
        printf(" %d\n", entrada);  
    else  
        break;  
}
```



Sentencia *break*

- El uso de `break` en un bucle no es recomendable, ya que puede hacer difícil la comprensión del comportamiento del programa. En particular, suele hacer muy difícil verificar los invariantes de los bucles. Por otra parte, suele ser fácil la escritura de los bucles sin la sentencia `break`.



Ejercicios

- Elaborar un programa que sume los números enteros positivos introducidos por el usuario desde la consola. El programa se detendrá cuando el usuario introduzca un -1.
- Dado un número N , calcular la suma de la serie $1 + 1/2 + 1/3 + \dots + 1/N$
- Dado un número N , calcular la suma de la serie $1 + 1/2! + 1/3! + \dots + 1/N!$
- Dados dos números n y x , calcular el valor de e^x que se puede aproximar por la suma: $e^x = 1 + x + x^2/2! + x^3/3! + \dots + x^n/N!$
- Recibir un entero N , e imprimir la tabla de multiplicación del número recibido.



Bibliografía

- Brian Kernighan; P. J. Plauger (1976), **Software Tools**, Addison-Wesley, pp. 352, ISBN 020103669X.
- Cairó Oswaldo. **Metodología de la programación- Algoritmos, diagramas de flujo y programas-**. Ed. Alfaomega.
- Joyanes Aguilar, Luis. **Fundamentos de programación**. Ed. Mc Graw Hill.
- Joyanes Aguilar, Luis. **Programación en C**. McGraw Hill.
- Schildt, Herbert. **Lenguaje C, Programación Avanzada**. McGraw Hill.
- Bronson, Gary. **Algorithm Development and Program Design Using C**, PWS Publishing Co.; Book and Disk edition (February 15, 1996) ISBN: 0314069879.
- Standish, Thomas. **Data Structures, Algorithms, and Software Principles in C**, Addison-Wesley Pub Co; 1st edition (September 30, 1994), ISBN: 0201591189.
- Linden, Peter. **Expert C Programming**, Prentice Hall, 1994, ISBN 0131774298.
- Reek, Kenneth. **Pointers on C**. Addison Wesley, 1997. ISBN 067399866.
- Kernighan, Brian W. & Dennis, M. Richie. **El lenguaje de programación C**. 2ª Ed. Prentice Hall, 1988.
- <http://zinjai.sourceforge.net/>