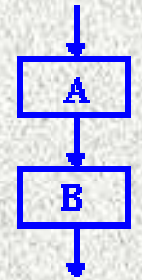
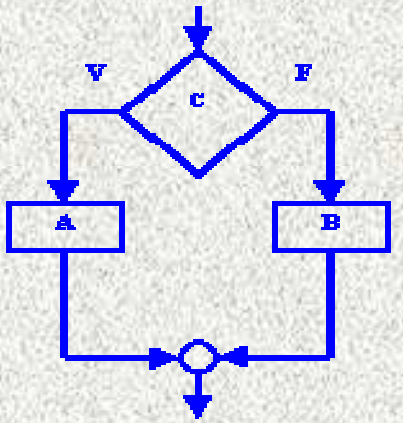
LENGUAJE C: ESTRUCTURAS DE CONTROL

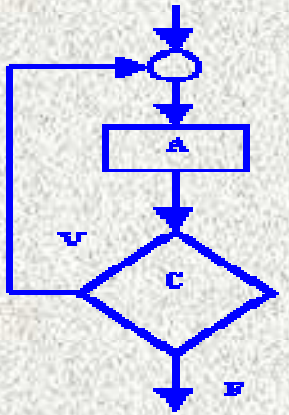
- **Secuencial**: las instrucciones se ejecutan una por una desde la primera hasta la última en el orden en que aparecen.



- **Selectiva**: en base a una condición u opción se realizan una o varias instrucciones. Las condiciones se especifican usando expresiones lógicas (***if***). Puede ser:
  - Simple
  - Doble
  - Múltiple



- **Repetitiva**: ejecuta iterativamente (en forma repetida) un bloque de instrucciones una o varias veces según el estado de un contador o el cumplimiento de una condición.



ESTRUCTURA SELECTIVA SIMPLE

- Están basadas en el cumplimiento de una condición. La estructura **si** (*condición*) – **entonces** (*acción*); evalúa la *condición* y si es **verdadera** ejecuta la *acción* (o bloque de acciones). Si la *condición* es **falsa**, no se hace nada.

- Pseudocódigo

```
si (condición)
    inicio
        inst_1
        inst_2
        . . .
        inst_n
    fin
```

Diagrama Flujo

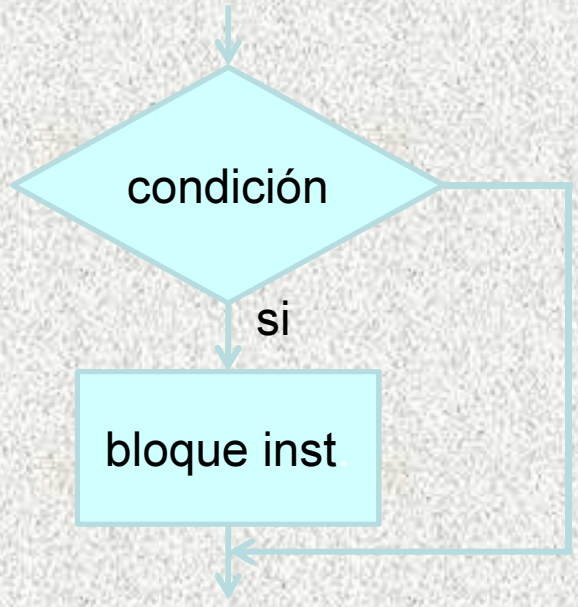
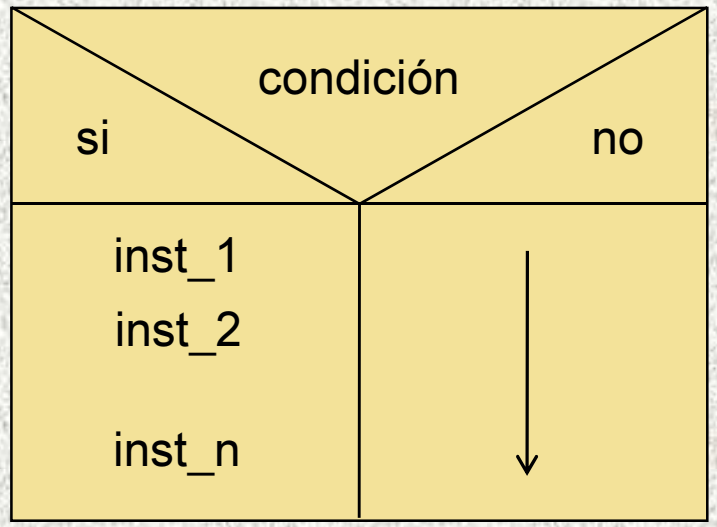


Diagrama Nassi-Schneiderman



- Sintaxis en C:

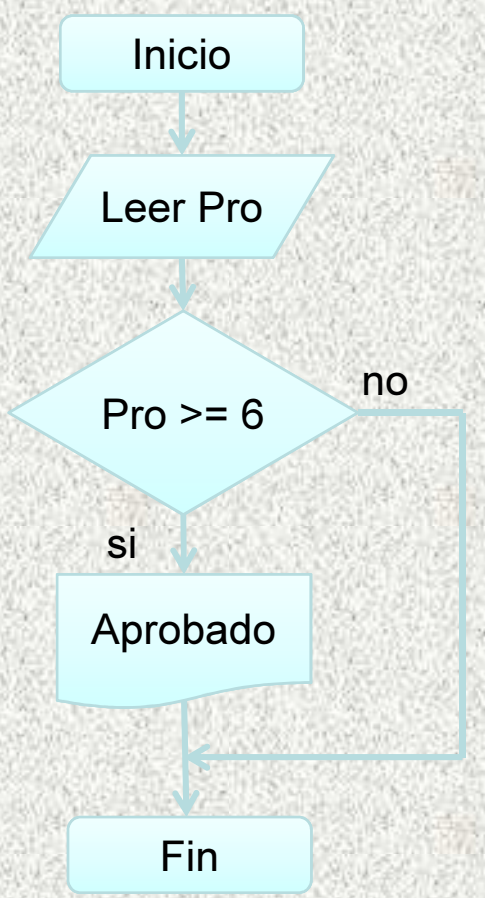
```
if (condición)
{
    inst_1;
    inst_n;
}
```

ESTRUCTURA IF: Ejemplo

- Problema: escribir un programa que al recibir el promedio de un alumno en un curso, escriba “Aprobado” si el promedio es mayor o igual a 6.
- Análisis:
  - **Entrada**: se recibe variable de tipo real que representa el promedio del alumno. Sea **Pro** esa variable.
  - **Salida**: se debe imprimir el mensaje “Aprobado” si se cumple la condición establecida.
  - **Proceso**: **si** se cumple la expresión lógica ‘el promedio es mayor o igual a 6’, **entonces** se imprimirá “Aprobado”.
- Verificación:

Corrida	Dato Pro	Resultado
1	6,75	“Aprobado”
2	5,90	
3	4,00	
4	8,80	“Aprobado”

Diagrama Flujo

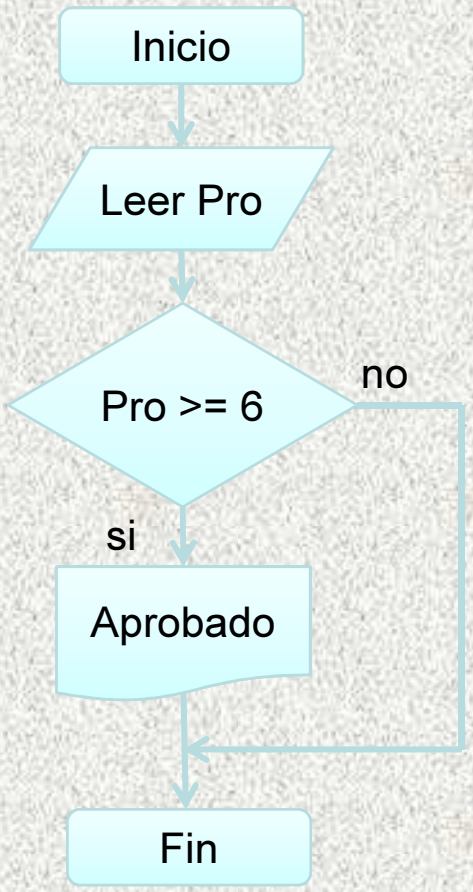


ESTRUCTURA IF: Ejemplo (continuación)

- **Codificación en Lenguaje C**:

```
/* El programa recibe como dato el promedio de un alumno y
   escribe “Aprobado” si el mismo es mayor o igual a 6          */

#include  <stdio.h>
#define   VALOR    6

int main()
{
   float Pro;

   printf(“Ingrese el promedio del curso: \n”);
   scanf("%f", &Pro);

   if (Pro >= VALOR)
   {
        printf(“\“Aprobado\”\n”, valor);
   }
   return 0;
}
```

Diagrama Flujo



ESTRUCTURA SELECTIVA DOBLE

- Se utilizan cuando se tienen dos opciones de acción mutuamente excluyentes. Se debe ejecutar una o la otra, pero no ambas a la vez. La estructura **si** (*condición*) – **entonces** (*acción1*) – **sino** (*acción2*); evalúa la *condición:* si es **verdadera** ejecuta la *acción1* (o bloque de acciones) y si es **falsa** se ejecuta *acción2*.

- Pseudocódigo

```
si (condición)
   inicio
      inst1_1
      inst1_2
       . . .
      inst1_n
   fin
sino
   inicio
      inst2_1
      inst2_2
       . . .
      inst2_n
   fin
```

Diagrama Flujo

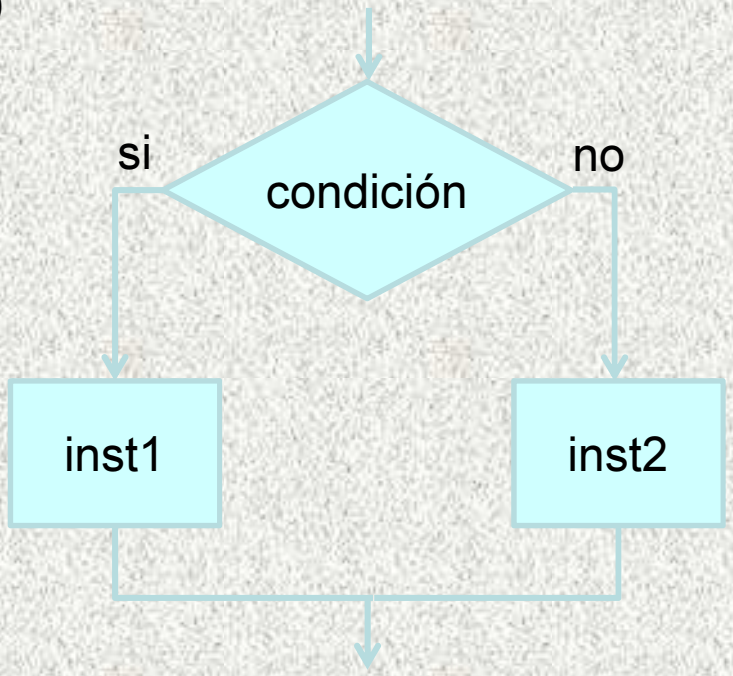
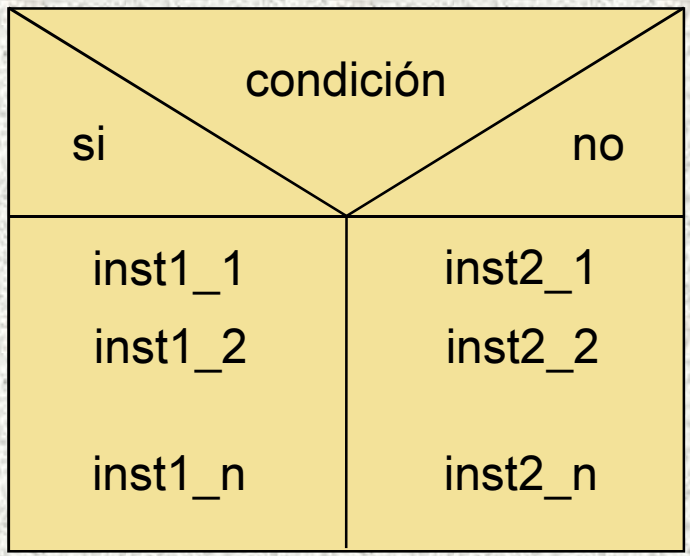


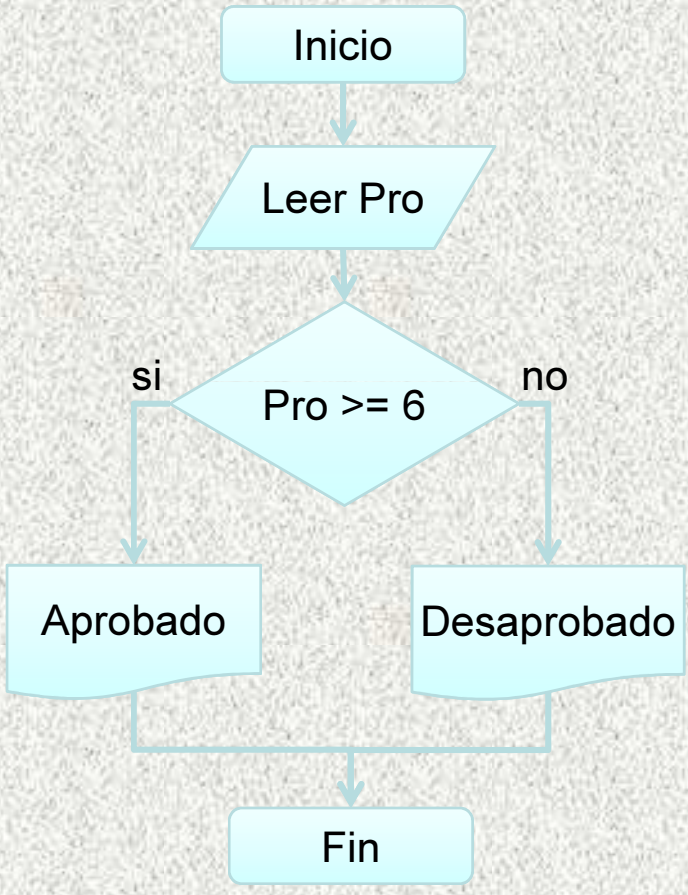
Diagrama Nassi-Schneiderman



ESTRUCTURA IF – ELSE

- Sintaxis en C:

```
if (condición)
{
    inst1_1;
    inst1_n;
}
else
{
    inst2_1;
    inst2_n;
}
```



- **Ejemplo**: modificar el ejemplo anterior para que imprima “Desaprobado” si Pro < 6.

```
    ⋮
if (Pro >= VALOR)
{
    printf("\"Aprobado\"\n", valor);
}
 else
{
    printf("\"Desprobado\"\n", valor);
}
    ⋮
```

OPERADOR CONDICIONAL '?'

- Sintaxis en C: (condición) **?** inst_1 **:** inst_2 **;**

  Si condición es V se ejecuta la instrucción inst_1; si es F se ejecuta inst_2. Se puede utilizar como alternativa a la estructura if/else. Las sentencias inst_1 e inst_2 deben ser simples, no pueden ser un bloque de instrucciones.

- **Ejemplo**: la estructura if/else usada en el ejemplo anterior

```
    ⋮
if (Pro >= VALOR)
{
    printf("\"Aprobado\"\n", valor);
}
 else
{
    printf("\"Desprobado\"\n", valor);
}
    ⋮
```

puede reemplazarse por :

```
(Pro >= VALOR) ? printf("\"Aprobado\"\n", valor) : printf("\"Desprobado\"\n", valor);
```

Ej. 01: if_else

ESTRUCTURAS IF ANIDADAS (NESTED)

- Representan ejecuciones alternativas mutuamente excluyentes.

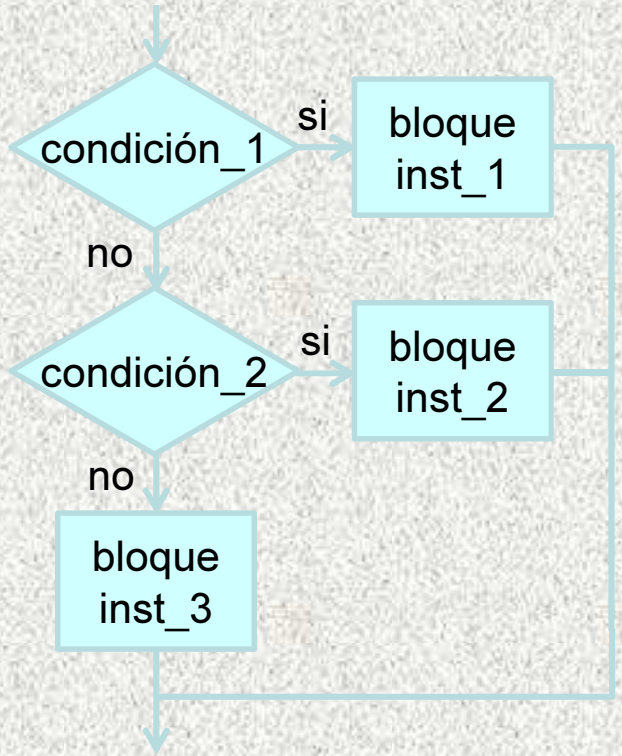
- Sintaxis en C:

```
if (condición _1)
{
    bloque_instrucciones_1;
}
else
    if (condición _2)
    {
        bloque_instrucciones_2;
    }
    else
    {
        bloque_instrucciones_3;
    }
```

Pseudocódigo:

```
si (condición_1)
  inicio
    acciones_1
  fin
sino
  si (condición_2)
    inicio
      acciones_2
    fin
  sino
    inicio
      acciones_3
    fin
```

Diagrama de Flujo:



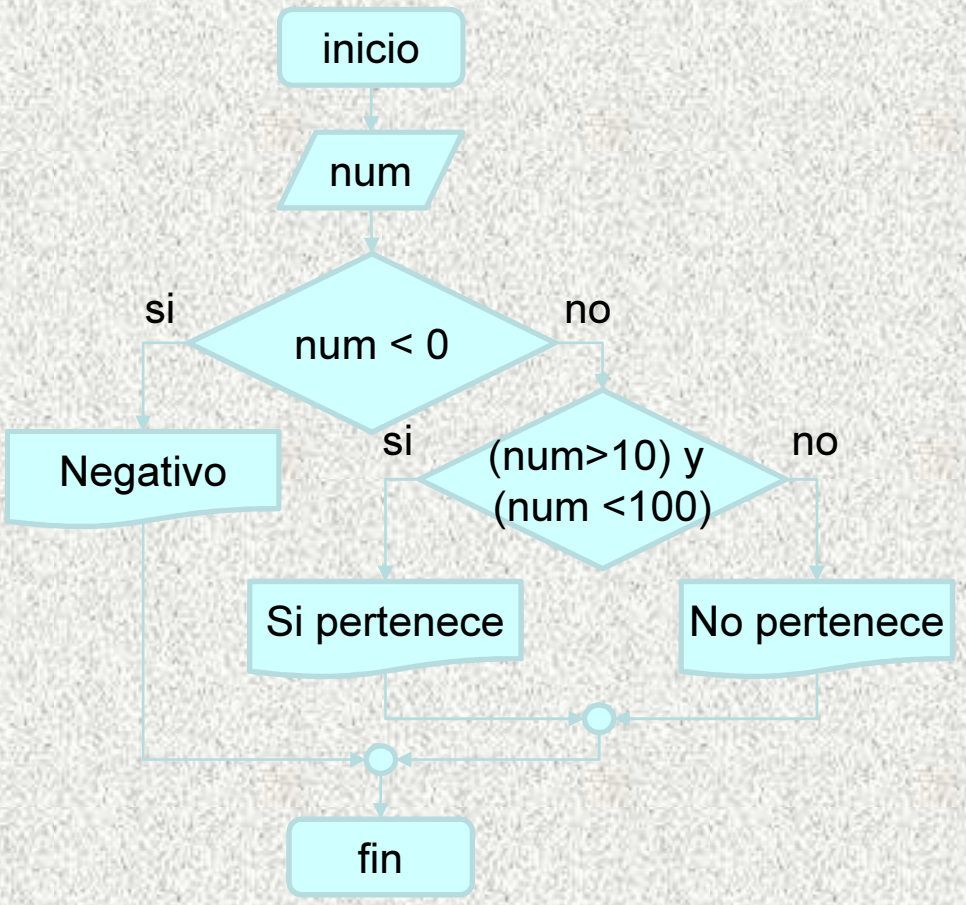
- Se pueden seguir anidando (nesting) estructuras if, if/else. Si el número de alternativas es grande la legibilidad del problema se hace más compleja y se dificulta el seguimiento del programa. Solución: estructura **switch**.

ESTRUCTURAS IF NESTED: Ejemplo

- Problema: escribir un programa que reciba un número entero, e imprima "Si pertenece" cuando se encuentre en el intervalo (10,100); ó "No pertenece" en caso contrario. Además debe dar un aviso si el número es negativo.

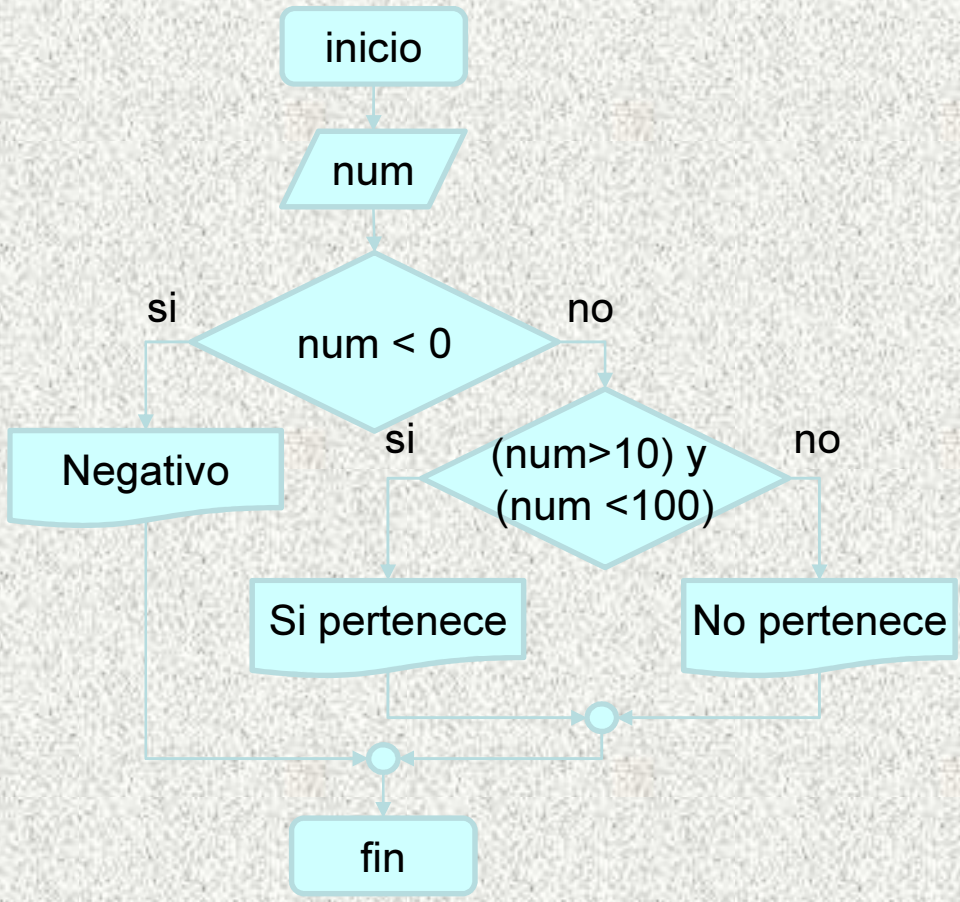
- Análisis:
  - **Entrada**: se recibe variable de tipo entera, num.
  - **Proceso**: se debe verificar que num es positiva y 10<num<100, e imprimir los mensajes.
  - **Salida**: se debe imprimir "Negativo" si num<0; "Si pertenece" si 10<num<100 y "No pertenece" en caso que num>0, pero se encuentre fuera del intervalo.

Diagrama Flujo



```
/****** Código en C ******/
#include  <stdio.h>
int main()
{
    int num;
    printf("\nIngrese un número: ");
    scanf("%i", &num);
    if (num<0)
    {
        printf("\nNegativo");
    }
    else
    {
        if( (num>10) && (num<100) )
        {
            printf("\nSi pertenece");
        }
        else
        {
            printf("\nNo pertenece");
        }
    }
    return 0;
}
```

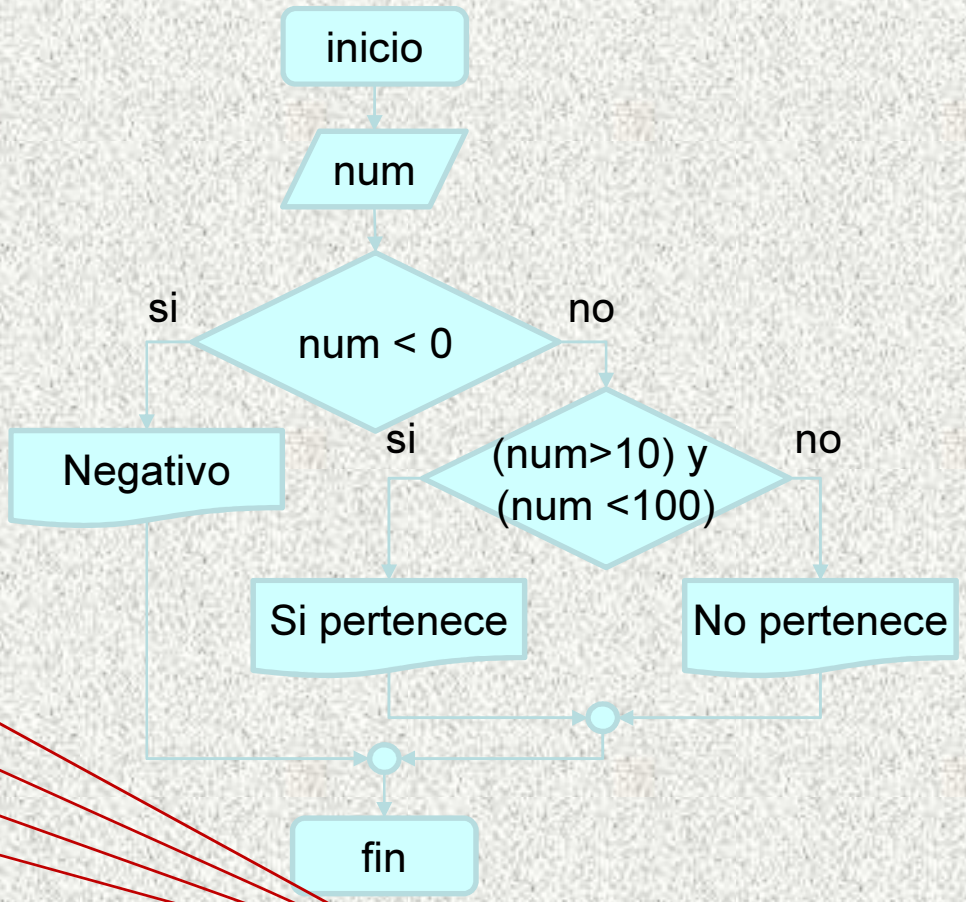
Diagrama Flujo



Ej. 02: ifelse_if

```
/****** Código en C ******/
#include  <stdio.h>
int main()
{
    int num;
    printf("\nIngrese un número: ");
    scanf("%i", &num);
    if (num<0)
    {
        printf("\nNegativo");
    }
    else
    {
        if( (num>10) && (num<100) )
        {
            printf("\nSi pertenece");
        }
        else
        {
            printf("\nNo pertenece");
        }
    }
    return 0;
}
```

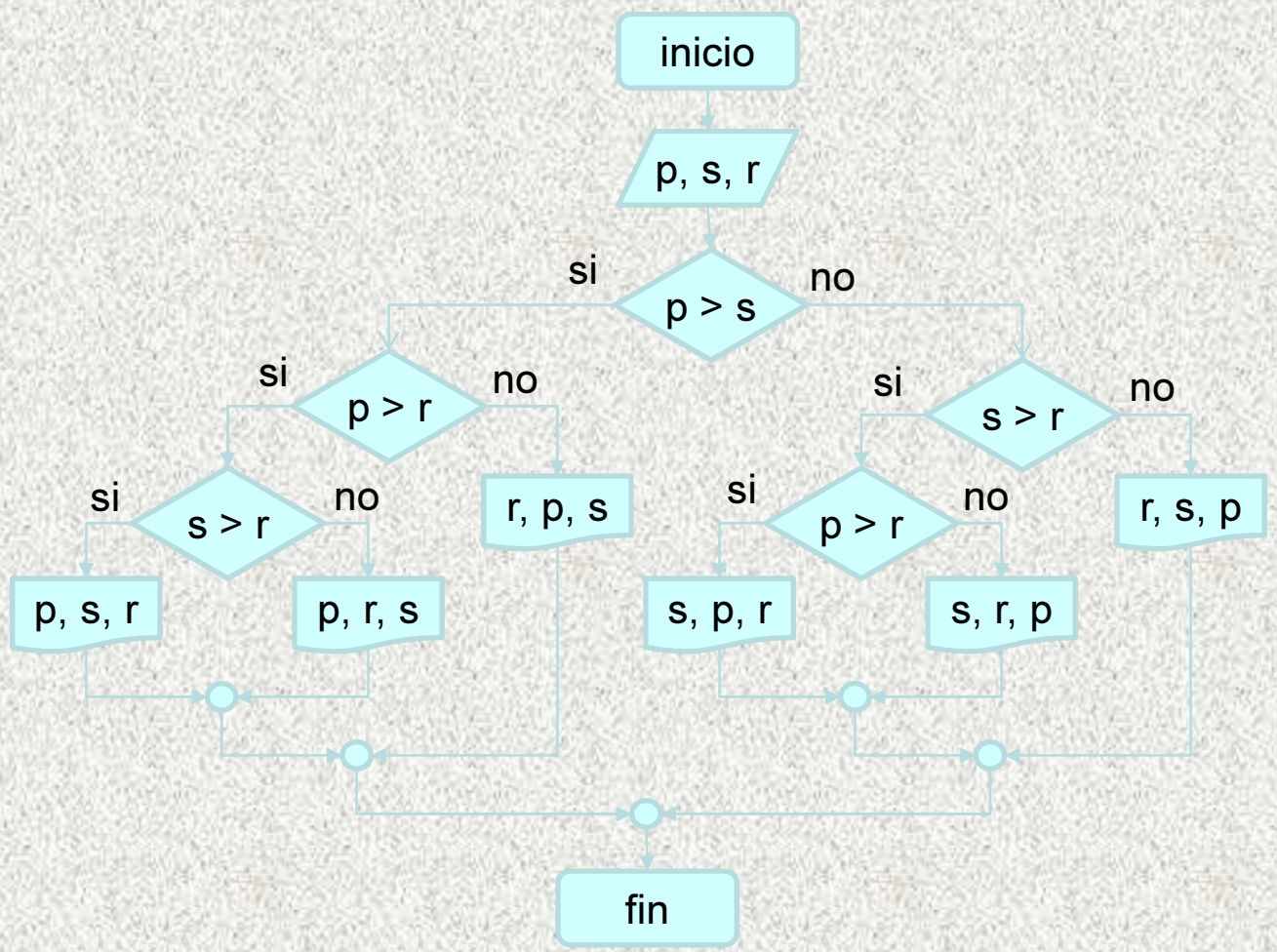
Diagrama Flujo



/* Se pueden eliminar las { } pero se dificulta la lectura del código, NO ES ACONSEJABLE y se desalienta fuertemente su omisión */

ESTRUCTURAS EN CASCADA: Ejemplo

- Problema: escribir un programa que lea tres valores de tipo flotante (p, s, r) y los imprima ordenados en forma descendente (de mayor a menor).



Ej. 03: if_cascada

ESTRUCTURA SELECTIVA MULTIPLE

- Se utilizan cuando se tienen varias opciones (***n***) de acción que pueden ser mutuamente excluyentes. En general se ejecuta sólo una de las *n* sentencias, siguiendo un determinado camino entre los *n* posibles del diagrama de flujo. Se facilita le legibilidad del programa y la búsqueda de errores.
- Se representa por un selector (**switch**) el cual ejecutará la sentencia 1 (o bloque de sentencias) si toma el valor 1, ejecutará la sentencia 2 si toma el valor 2, etc.
- La decisión **no se basa en una condición**, sino **en el valor de una variable** (el parámetro de **switch**), que sólo puede ser de tipo **int** o **char**.
- Al finalizar cada bloque de sentencias se debe incluir la instrucción **break**, que provoca la finalización de la ejecución de la selección.
- Si el valor de la variable selectora no coincide con el valor de alguno de los bloques se ejecuta (si existe) el bloque por defecto (**default**).

ESTRUCTURA SELECTIVA MULTIPLE

➢ Pseudocódigo

```
según_sea variable
    inicio
        caso valor1:
            inst_1
            salir
        caso valor2:
            inst_2
            salir
        . . .
        caso valorN:
            inst_N
            salir
        caso contrario:
            inst_defecto
    fin
```

Diagrama Flujo

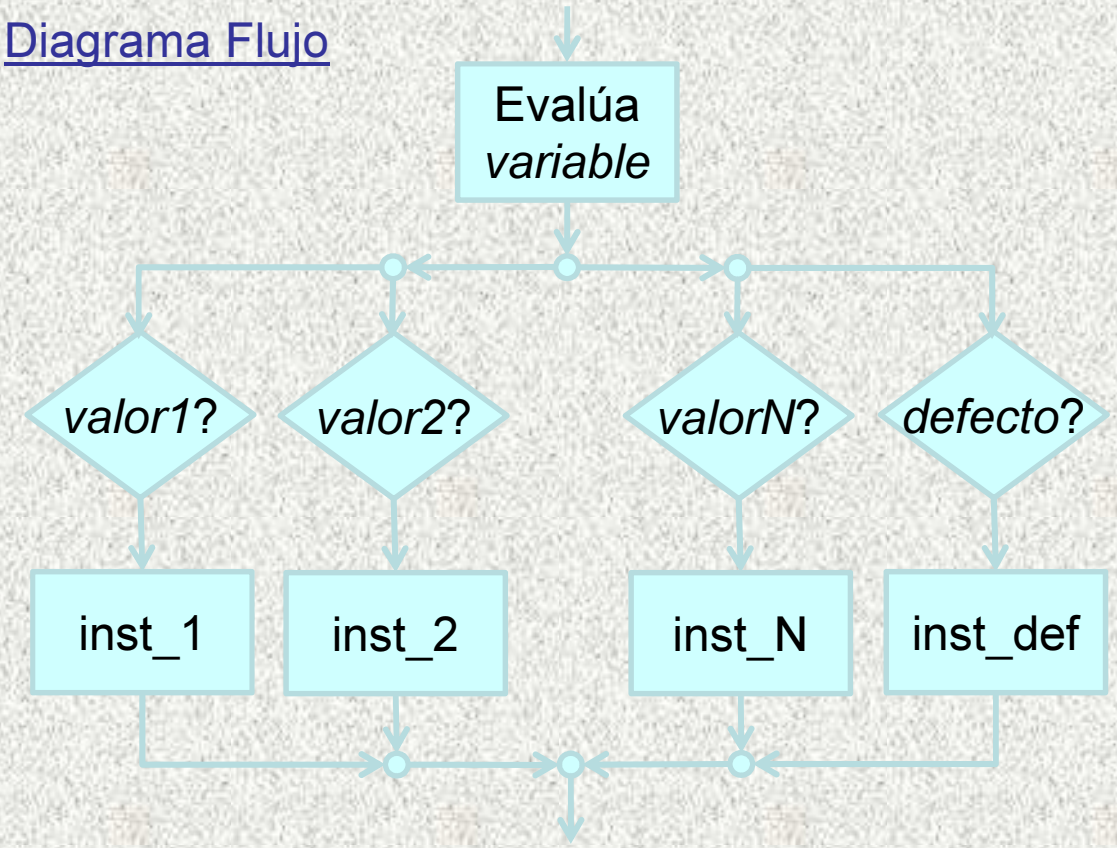
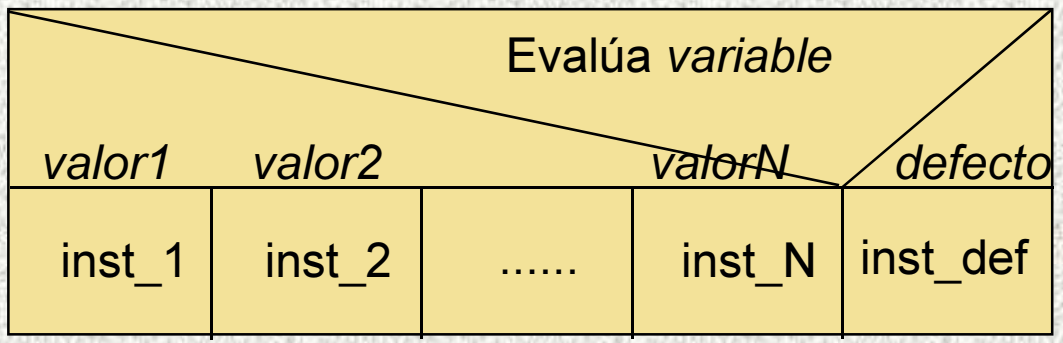


Diagrama Nassi-Schneiderman



ESTRUCTURA SWITCH EN C

```
switch (variable)
{
    case valor1:
        inst_1;
        break;          /* si se omite continúa ejecutando el código (case valor2)
    case valor2:
        inst_2;
        break;
        ⋮
    case valorN:
        inst_N;
        break;
    default :
        inst_def        /* no es necesario el break */
}
```

COMPARACION SWITCH – IF ANIDADOS

```
#include  <stdio.h>
int main()
{
    int num;
    printf(“\nIngrese un número: ”);
    scanf(“%i", &num);
    switch (num)
    {
        case 1:
            printf(“Es un 1\n”);
            break;
        case 2:
            printf(“Es un 2\n”);
            break;
        case 3:
            printf(“Es un 3\n”);
            break;
        default:
            printf(“No es ni 1 ni 2 ni 3\n”);
    }
    return 0;
}
```

```
#include  <stdio.h>
int main()
{
    int num;
    printf(“\nIngrese un número: ”);
    scanf(“%i", &num);
    if (num==1)
        printf(“Es un 1\n”);
    else
        if(num==2)
            printf(“Es un 2\n”);
        else
            if(num==3)
                printf(“Es un 3\n”);
            else
                printf(“No es ni 1 ni 2 ni 3\n”);
    return 0;
}

/* faltan las {} para legibilidad = ERROR! */
```

SWITCH: Ejemplo

Problema: escribir un programa que reciba el número del mes (1-12) y determine la cantidad de días del mismo. Suponer año bisiesto.

```
#include  <stdio.h>
#define    BISIESTO     1
int main()
{
    int mes, dias;
    printf("\nIngrese el número del mes: ");
    scanf("%i", &mes);
    switch (mes)
    {
        case 4: case 6: case 8: case 11:
            dias = 30;
            break;
        case 2:
            if(BISIESTO)
                dias = 29;
            else
                dias = 28;
            break;
        default:
            dias = 31;
    }
    printf("\n\n El mes %d tiene %d dias.", mes,  dias);
    return 0;
}
```

ESTRUCTURAS REPETITIVAS

- Permiten repetir (iterar) la ejecución de un bloque de instrucciones de acuerdo al cumplimiento de una determinada condición. Los valores de los datos pueden variar con cada iteración.
- Todo ciclo debe terminar (número finito de repeticiones) por lo que debe existir una acción de Parada o Fin del Ciclo. En cada iteración, se evalúan las condiciones para decidir si se debe seguir o se debe detener.
- **for**: se utiliza generalmente cuando se conoce a priori la cantidad de iteraciones necesarias, que se carga en un contador. Contiene una instrucción de **inicialización** y otra de **actualización**.
- **while**: repite las instrucciones mientras una condición sea cierta. Se usa cuando no se conoce el número de iteraciones.
- **do while**: similar a los anteriores, con la diferencia de que la condición se evalúa al final del bloque; ejecutando las sentencias al menos una vez sin importar la condición. Se utiliza principalmente para validar los datos de entrada.

ESTRUCTURAS REPETITIVAS

➢ Pseudocódigo:

for (para)

para *contador*
  desde *Valor_inicial*
  hasta *Valor_final*
  [**paso** = *Incremento*]
  hacer
    inst_1
    . . .
    inst_N
fin_para

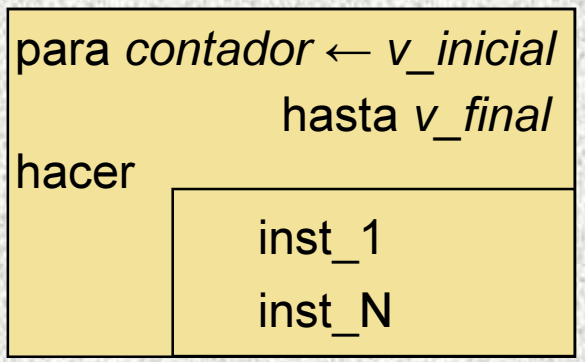
while (mientras)

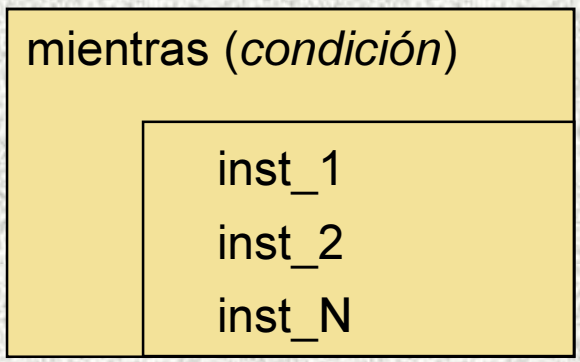
mientras *condición* **hacer**
  inst_1
  inst_2
  . . .
  inst_N
fin_mientras

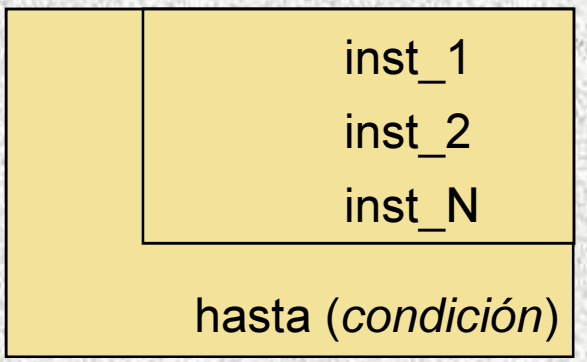
do while (hacer mientras)

hacer
  inst_1
  inst_2
  . . .
  inst_N
mientras_que *condición*

➢ Diagrama de Chapin (N-S):



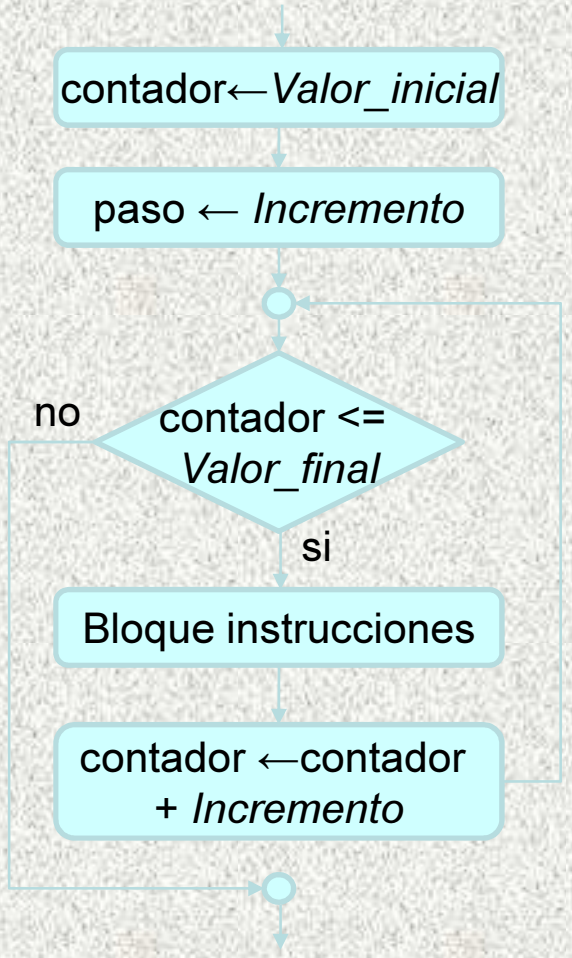




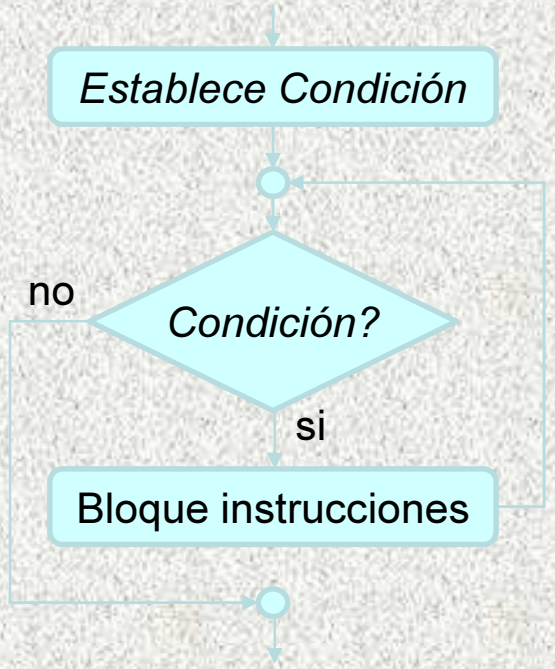
ESTRUCTURAS REPETITIVAS

- Diagrama de flujo:

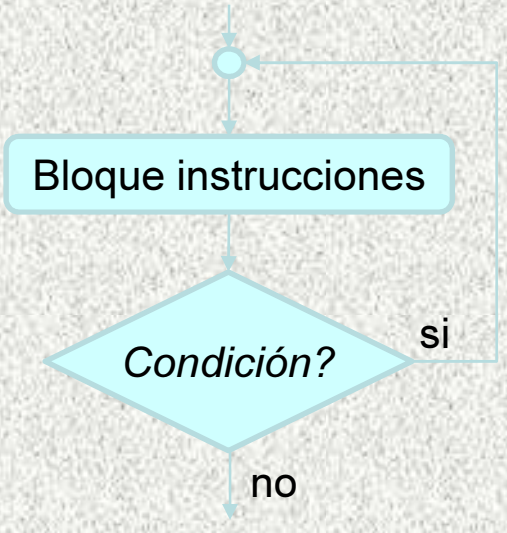
for (para)



while (mientras)



do while (hacer mientras)



ESTRUCTURA FOR EN C

- Sintaxis:

inicialización expresión de control actualización

```
for(contador=valor_inicial; contador <= valor_final; contador++)
{
    bloque_de_instrucciones;
}
```

- En la primera iteración se ejecuta la inicialización de la variable de control. Mientras la expresión de control sea cierta se ejecuta el bloque de instrucciones. Al finalizar el mismo se ejecuta la instrucción de actualización.
- Se pueden inicializar y actualizar más de una variable simultáneamente. En algunos casos puede faltar alguno de los campos (no se aconseja).
- Ejemplos:

```
for (cont=1; cont <=100; cont++)          /* Imprime los números  del 1 al 100 */
    printf( “Voy por el %i \n", cont );

for (cont=0, acum=0; cont <=20; cont+=2)          /* Suma en paso de a 2 */
    acum += cont;                                 /* los primeros 20 números enteros */
```

ESTRUCTURA WHILE EN C

- Sintaxis:

```
condicion = valor_inicial;
while (condicion)
{
   sentencia1;
   sentencia2;
     ….
   sentencia_modifica_condicion;
}
```

- Ejemplo:

```
int  i=0,
     ac=0;
while (i<100)                    /* suma los primeros 100 números 0-99 */
{
     printf("%d \n", i*i);
     ac+=i;
     i++;                        /* permite modificar el valor de i para condición */
}
```

ESTRUCTURA DO – WHILE EN C

- Imprime una tabla de multiplicación:

```
condicion = valor_inicial;
do                          /* condición se evalúa al final → se ejecuta al menos 1 vez */
{
   sentencia1;
   sentencia2;
     ….
   sentencia_modifica_condicion;
} while (condicion);                    /* verificar ; al final !! */
```

- Ejemplo:

```
int  i=0,
     ac=0;
do                                  /* suma los primeros 100 números 0-99 */
{
     printf("%d  %d\n", i, i*i);
     ac += i;
     i++;                           /* permite modificar el valor de i para condición */
} while (i < 100);
```

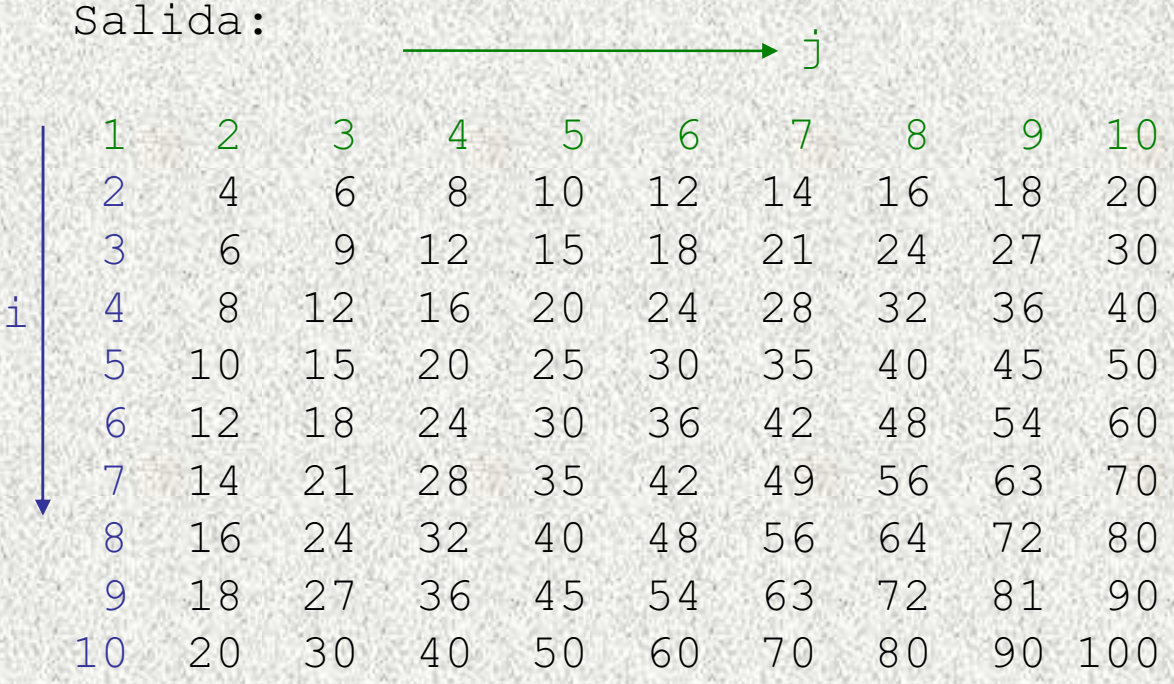
DO – WHILE: Ejemplo

```
/* Pide continuamente al usuario que ingrese un número entero entre 1 y 10.
   Si no cumple con la condición  sale del bucle */

 #include <stdio.h>

 int main()
{
     int n, error;

     do
     {
          printf("\nIngrese un número entero entre 1 y 10: ");
          scanf("%d", &n);

          error = (n < 1 || n > 10);
          if (error)
               printf("\n ERROR: Intentelo nuevamente!! \n");
     } while (error);

   /* ahora se asegura que n está en el rango de valores deseados */
    …   /* sentencias hasta el fin del programa */
```

BUCLES ANIDADOS: Ejemplo

```
#include <stdio.h>

int main()
{
    int  i=1;
    int  j;

    while(i <= 10)
    {
        j = 1;

        while(j <= 10)
        {
            printf(" %3d", i * j);
            j++;
        }

        printf("\n");
        i++;
    }

    return(0);
}
```

Salida:




```
/* imprime las 10 columnas de cada fila */

/* end while interno */

/* terminó una fila */

/* end while externo; terminó la tabla */

/* fin del programa */
```

Ejs. 06 y 07: bucle_anidado

SENTENCIA BREAK

- Permite alterar el flujo normal de una estructura repetitiva. Se utiliza generalmente en la estructura **switch** para asegurar que se ejecute sólo una de las opciones **case**. Además permite la finalización de los bucles **for**, **while** y **do while** (por ejemplo permite fijar un número límite de iteraciones).

```
for (ini;cond;actual)
{
   sentencias;
   if (condicion_salida)
   {
      break;
   }
   sentencias;
}
sentencias;
```

```
while (condicion)
{
   sentencias;
   if (condicion_salida)
   {
      break;
   }
   sentencias;
}
sentencias;
```

```
do
{
   sentencias;
   if (condicion_salida)
   {
      break;
   }
   sentencias;
} while (condicion);
sentencias;
```

SENTENCIA BREAK: Ejemplo

/* El programa realiza la sumatoria de los números ingresados, finaliza si uno < 0 */

```
#include <stdio.h>
int main()
{
    int  i;
    float num, suma=0;
    for (i=1; i<=10; i++)
    {
        printf("\nEntre número %d: ",i);
        scanf("%f", &num);
        if (num<0.0)
        {
            break;                          /* sale del for() si num < 0 */
        }                                                  /* end if()  */
        suma+=num;
    }                                                      /* end for() */
    printf("\nPromedio = %f", suma/(i-1) );
    return 0;
}
```

SENTENCIA CONTINUE

- También permite alterar el flujo normal de una estructura repetitiva. A diferencia de **break**, se pasa el control al inicio del bucle, evitando la ejecución de las sentencias que se encuentran a continuación.

```
for (ini;cond;actual)
{
    sentencias;
    if (condicion_salida)
    {
        continue;
    }
    sentencias;
}
sentencias;
```

```
while (condicion)
{
    sentencias;
    if (condicion_salida)
    {
        continue;
    }
    sentencias;
}
sentencias;
```

```
do
{
    sentencias;
    if (condicion_salida)
    {
        continue;
    }
    sentencias;
} while (condicion);
sentencias;
```

SENTENCIA CONTINUE: Ejemplo

```
/* El programa realiza la productoria de los 10 números ingresados y evita un 0 */

#include <stdio.h>

int main()
{
    int  i, num, producto;

    for (i=1, producto=1;  i<=10; i++)
    {
        printf("\nEntre número %d: ",i);
        scanf("%d", &num);

        if (num==0)
        {
              continue;                          /* evita multiplicar por 0 */
        }                                                  /* end if()  */

        producto*=num;
    }                                                      /* end for() */

    printf("\nProducto = %d", producto);
    return 0;
}
```

Ej. 08: continue