UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

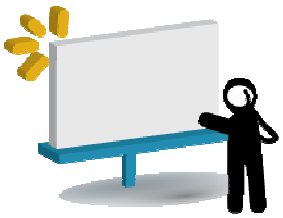
FUNDAMENTOS DE LA PROGRAMACIÓN

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

MÓDULO 2 - UNIDAD 5

Programación Estructurada

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning



Presentación:

En esta Unidad estaremos comenzamos con el segundo Módulo del curso, por lo que comenzaremos a ver las distintas formas de representar un algoritmo o programa, aunque centrándonos en el seudocódigo.

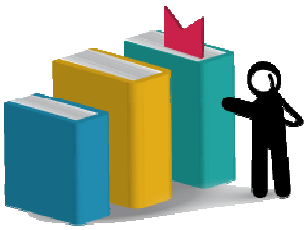
También tendremos la oportunidad de analizar las estructuras de control principales: la estructura secuencia, la estructura condicionales y la estructura repetitivas y las variantes de cada una.

Adicionalmente, analizaremos otras estructuras de uso común que nos acompañarán en el resto del curso.

Objetivos:

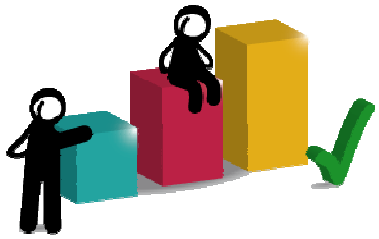
Que los participantes:

- Comprendan los principales aspectos del paradigma estructurado de programación
- Conozcan las estructuras de control y sus principales variantes
- Conozcan otras estructuras comunes



Bloques temáticos:

1. Programación estructurada
    - 1.1 Introducción
    - 1.2 Modularización
2. Detalles de estructuras básicas de control
    - 2.1 Estructura secuencial
    - 2.2 Estructura condicional o selectiva
    - 2.3 Estructura repetitiva o iterativa
3. Tipos de estructuras condicionales
    - 3.1 Condición simple
    - 3.2 Condición doble
    - 3.3 Condición compuesta
    - 3.4 Condición anidada
    - 3.5 Condiciones múltiples
4. Estructuras comunes
    - 4.1 Asignaciones
    - 4.2 Contadores
    - 4.3 Acumuladores



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

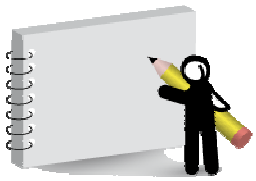
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

* El MEC es el modelo de E-learning colaborativo de nuestro Centro.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Programación estructurada

1.1 Introducción

Los lenguajes de programación basados en el paradigma estructurado tienen como fundamento el teorema de Dijkstra, desarrollado en los años '60, que posteriormente permitió demostrar que cualquier programa puede escribirse utilizando tres estructuras básicas de control:

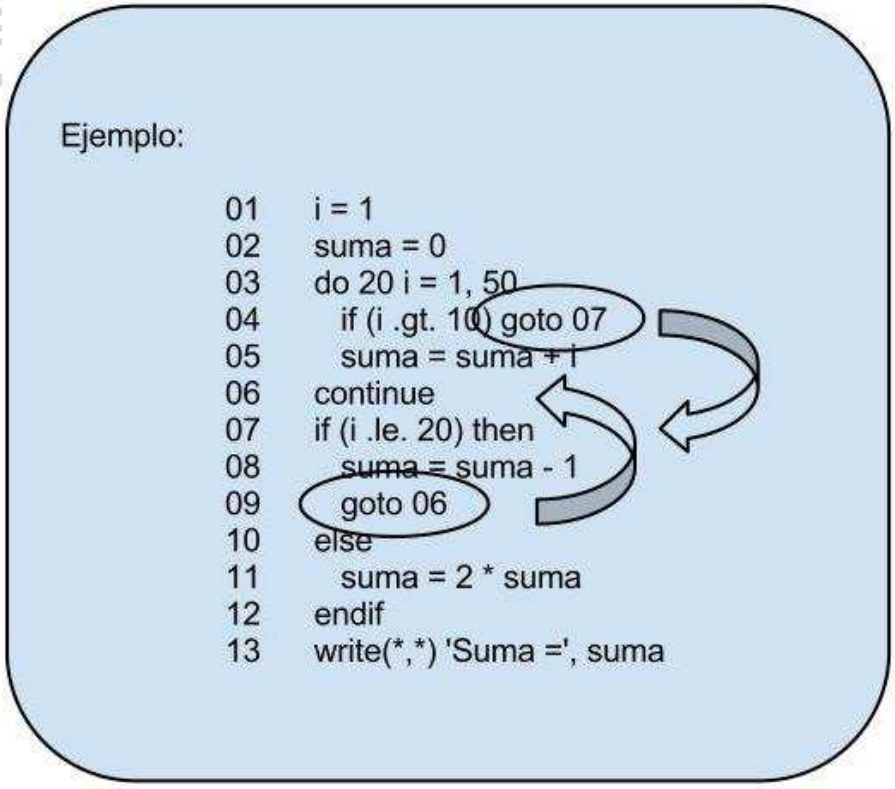
- **Secuencia**: tal como lo venimos viendo, una instrucción se ejecuta una detrás de otra, siendo este orden conocido

- **Condición**: es la posibilidad de crear bifurcaciones en los flujos de ejecución según se dé o no una condición determinada

- **Repetición**: es la característica que permite volver a ejecutar una o varias instrucciones una cantidad de veces determinada

Los programas escritos utilizando sólo estas tres instrucciones de control básicas y sus variantes, evitando la instrucción **goto** ("ir a" en inglés), se definen como **Estructurados**.

Esta instrucción (goto) era muy común en lenguajes antiguos (como Fortran o Basic) y permitía modificar el flujo de un problema (su secuencia) dando "saltos" dentro de un programa. La idea básica es que cada línea o instrucción del programa se encontraba numerada, por lo que llegado el momento de utilizar la palabra reservada goto (o "go to" dependiendo del lenguaje) se utilizaba esta instrucción indicando a continuación el número de línea en la cual continuaba el programa.



La definición común sobre la programación estructurada hace referencia a la forma en que se controla y produce la ejecución de las instrucciones del programa. La norma general es que las instrucciones se ejecuten sucesivamente una detrás de otra (secuencia), aunque diferentes partes del programa se vayan a ejecutar o no dependiendo del cumplimiento de alguna situación (condición). Asimismo, hay instrucciones que pueden ejecutarse varias veces (repetición), ya sea en número predeterminado o hasta que se cumpla una condición dada.

A finales de los años sesenta, con la difusión del teorema antes mencionado, comenzó a surgir un nuevo paradigma de programación, que tiende a reducir a la mínima expresión el uso de la instrucción goto y la sustituye por otras más simples de usar y entender. Por dicho motivo, a esta forma de programar (la Programación Estructurada) se la conoce como “programación sin goto”.

1.2 Modularización

Otra de las visiones en cuanto al concepto de programación estructurada tiende a referirse a la subdivisión de un programa en partes más pequeñas, claras y simples, normalmente conocidas como segmentos o módulos, evitando de esta forma la creación de programas extensos, complejos y probablemente poco entendibles (por ende, costosos en cuanto al esfuerzo requerido para modificarlos o mejorarlos).

Con el tiempo fueron surgiendo ciertas reglas o normas tendientes a darle ciertos parámetros de calidad a la programación, por ejemplo, al definir que cada módulo del programa no exceda, en longitud, de una página de codificación, o sea, alrededor de 50 líneas (50 instrucciones), aunque esta parámetro rara vez se cumple.

De esta forma, podemos decir que un programa estructurado está compuesto por módulos, los cuales están conformados por un conjunto acotado de instrucciones y código. Cada una de estas partes contendrá funciones y datos relacionados semántica y funcionalmente. En una correcta partición del programa deberá resultar fácil e intuitivo comprender lo que hace cada módulo.

En una modularización correcta, la comunicación entre módulos se lleva a cabo de una manera cuidadosamente controlada. Asimismo, una correcta partición del problema producirá una nula o casi nula dependencia entre los módulos, pudiéndose entonces trabajar con cada uno de estos módulos de forma independiente. Este hecho garantizará que los cambios que se efectúen a una parte del programa, durante la programación original o su mantenimiento, no afecten al resto del programa que no ha sufrido cambios.

Esta técnica de programación presenta las siguientes ventajas principales:

a) El hecho de subdividir un programa facilita **desintegrar un problema complejo** y atacar cada una de sus partes componentes, en lugar de tener que abordar el problema completo

b) Permite **paralelizar trabajo entre varios grupos de programadores**, reduciendo el tiempo de programación de los proyectos

c) Posibilita en mayor grado **la reutilización del código** en futuros proyectos, en caso de tener los módulos correctamente atomizados y encapsulados

ACTIVIDAD 1:

¿Porqué la paralelización del trabajo implica reducir los tiempos de programación? ¿En qué medida se reducen? ¿Qué condiciones deberían darse para que esto suceda... o sucede siempre?

Y ¿porqué es importante la reutilización de código?

Reflexione sobre estos tópicos, complemente su conclusión investigando en Internet y comparta y justifique sus respuestas en el foro de actividades de la Unidad.

2. Detalles de estructuras básicas de control

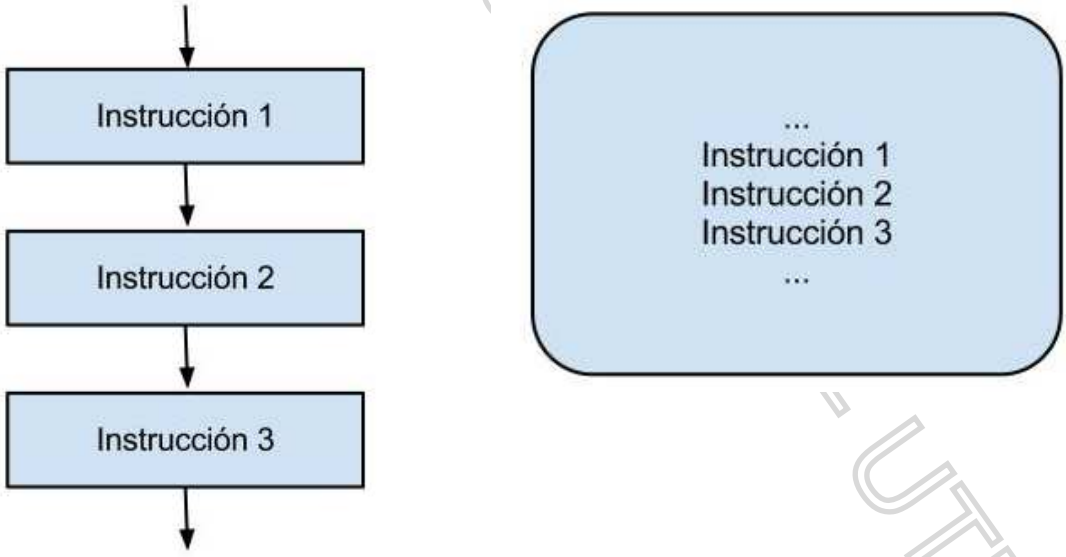
Como se dijo anteriormente, la programación estructurada tiene como principales características:

- Una forma de organización **modularizada**
- **Tres estructuras** básicas de control

Sobre este segundo aspecto, brevemente mencionado anteriormente, detallaremos su aplicación a continuación.

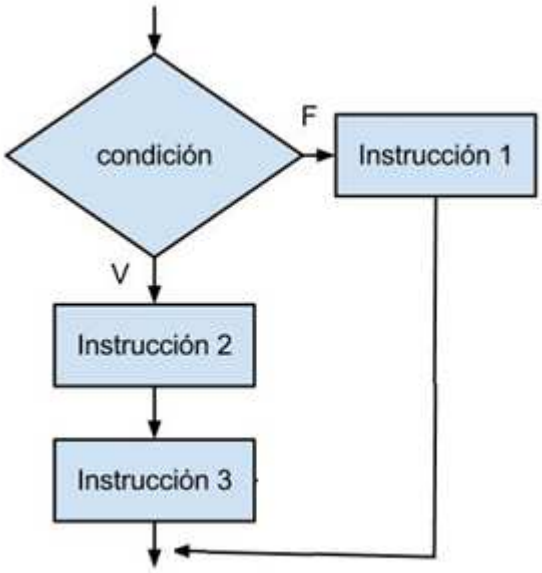
2.1 Estructura secuencial

La **secuencialidad** significa que las instrucciones de un programa se ejecutan una después de la otra, en el mismo orden en el cual aparecen en el programa, sin saltos entre las instrucciones dentro del programa.



2.2 Estructura condicional o selectiva (bifurcación)

Esta estructura plantea la **selección entre dos alternativas** (la "verdadera" o la "falsa") basándose en el resultado de una condición (pregunta o cuestionamiento).




```
si <<condición>> entonces
    Instrucción 2
    Instrucción 3
sino
    Instrucción 1
fin si
```

En este ejemplo, se evalúa una <<condición>> en el símbolo de rombo. En caso en que ésta se cumpla (sea "verdadera") se ejecutan las instrucciones “Instrucción 2” e “Instrucción 3”. Notar la letra **“V” (de verdadero) en el flujo descendente desde el rombo**, que indica que ésta es la bifurcación que se ejecuta en caso de cumplirse la condición. En el caso de que la <<condición>> sea falsa (notar nuevamente **la letra “F” que indica esta opción de flujo**), se ejecuta la instrucción “Instrucción 1”.

Este tipo de estructura tiene la posibilidad de modificarse aumentando las prestaciones y complejidad, por lo que será desarrollada con mayor amplitud en esta misma Unidad.

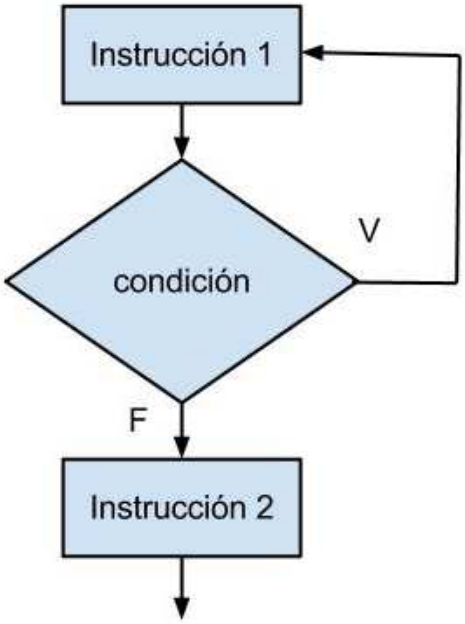
Ejemplos de <<condición>> podrían ser:

- a > 10 (donde "a" debería tener un valor numérico asignado)
- 5 <= numero2
- i = j
- 2 > 100 (sin mucho sentido, porque se conoce previamente el resultado y el camino que tomará el flujo del programa en la bifurcación)
- a <> b
- e <> 6

¡Recordar los operadores relacionales (lógicos) vistos en la Unidad anterior!

2.3 Estructura repetitiva o iterativa

Esta estructura permite la ejecución recursiva de una instrucción mientras se cumpla con una determinada condición.




```
...
Instrucción 1
mientras <<condición>>
   Intrucción 1
fin mientras
Instrucción 2
...
```

En el ejemplo, se ejecuta la Instrucción 1 por primera vez, y luego se vuelve a ejecutar una y otra vez **hasta que la <<condición>> sea falsa o, dicho de otra manera, se va a ejecutar el ciclo ("bucle") mientras la <<condición>> sea verdadera**.

Notar que en este caso se invirtieron las salidas del flujo condicional, ya que en el diagrama sale hacia la derecha la verdadera y hacia abajo, la falsa. Por esta razón resulta necesario aclarar en los diagramas cuáles son las opciones verdadera o falsa utilizando la letra "V" o "F" respectivamente, según corresponda.

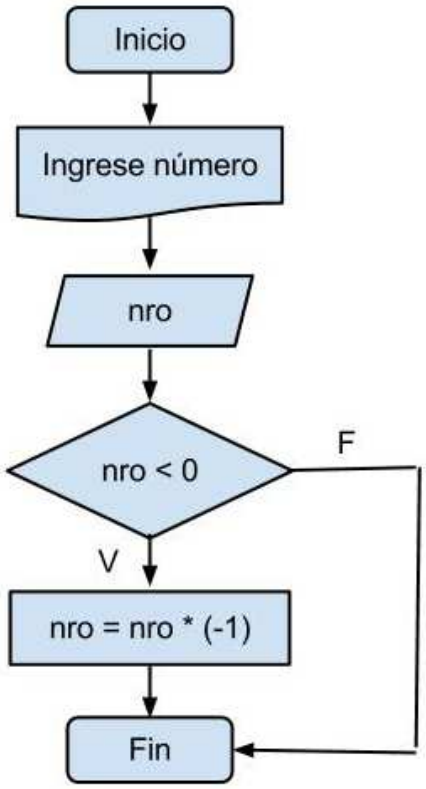
3. Tipos de estructuras condicionales

Como se dijo anteriormente, existen varios tipos de estructuras condicionales, según las necesidades puntuales que surjan en casa caso.

Todas estas estructuras se encuentran implementadas en los lenguajes de programación modernos.

3.1 Condición simple

Éste es el tipo de estructura más simple, en la que el contenido de la condición (instrucciones dentro del bloque de la estructura condicional) se ejecuta solamente para el cumplimiento de la condición (si es verdadera).




```
programa numeroPositivo

inicio
    mostrar: "Ingrese número"
    ingresar: nro
    si nro < 0 entonces
        nro = nro * (-1)
    fin si
fin
```

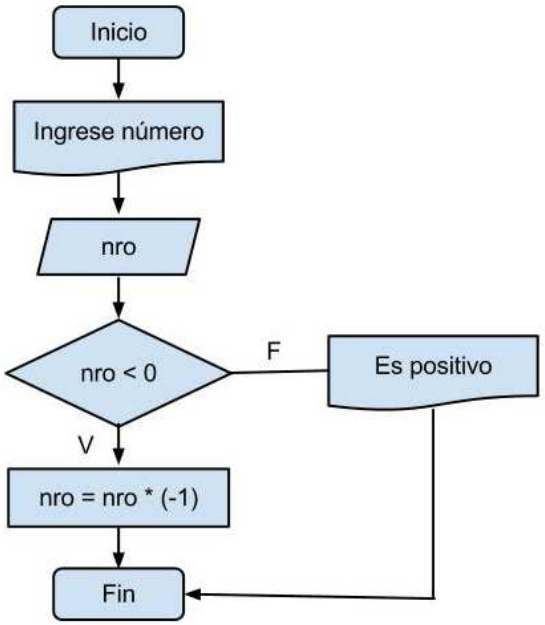
En este ejemplo, podemos ver que el programa solicita el ingreso de un número. Posteriormente, la estructura condicional verifica si el número es negativo (preguntando si "nro" es menos a "0" -cero-). En caso de cumplirse con la condición, el flujo del programa ingresa en la estructura condicional, donde se toma el número ingresado y se transforma en positivo multiplicándolo por -1. Posteriormente finaliza el programa.

En caso de haberse ingresado originalmente un número positivo, el programa simplemente lo verifica en la estructura condicional y termina el programa sin realizar ninguna otra actividad.

3.2 Condición doble

Esta estructura es similar a la anterior, con la diferencia de que se contemplan la **posibilidad de ocurrencia de ambas opciones de la condición (una u otra, nunca ambas a la vez)**: si se cumple, se ejecuta una serie de instrucciones. Si no se cumple, se ejecutan otras, como veremos a continuación, siguiendo con el ejemplo antes visto.

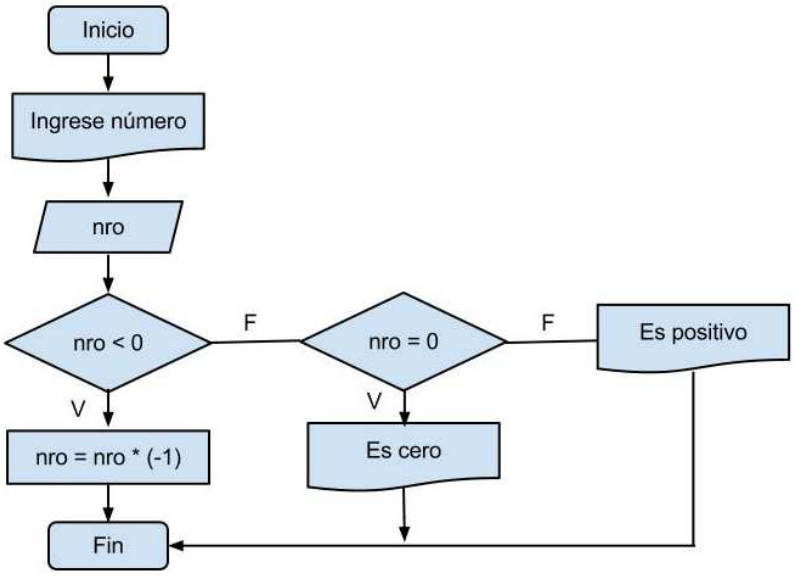
Aquí se destaca la **palabra reservada "sino"** en el seudocódigo, que indica el flujo alternativo que se ejecuta en caso de no cumplirse la condición dada.




```
programa numeroPositivo

inicio
    mostrar: "Ingrese número"
    ingresar: nro
    si nro < 0 entonces
        nro = nro * (-1)
    sino
        mostrar: "Es positivo"
    fin si
fin
```

3.3 Condición compuesta

Si bien esta estructura tiende a quedar en desuso debido a que se la considera confusa y poco intuitiva, cabe destacarla ya que sigue siendo relativamente común encontrarla en programas existentes.



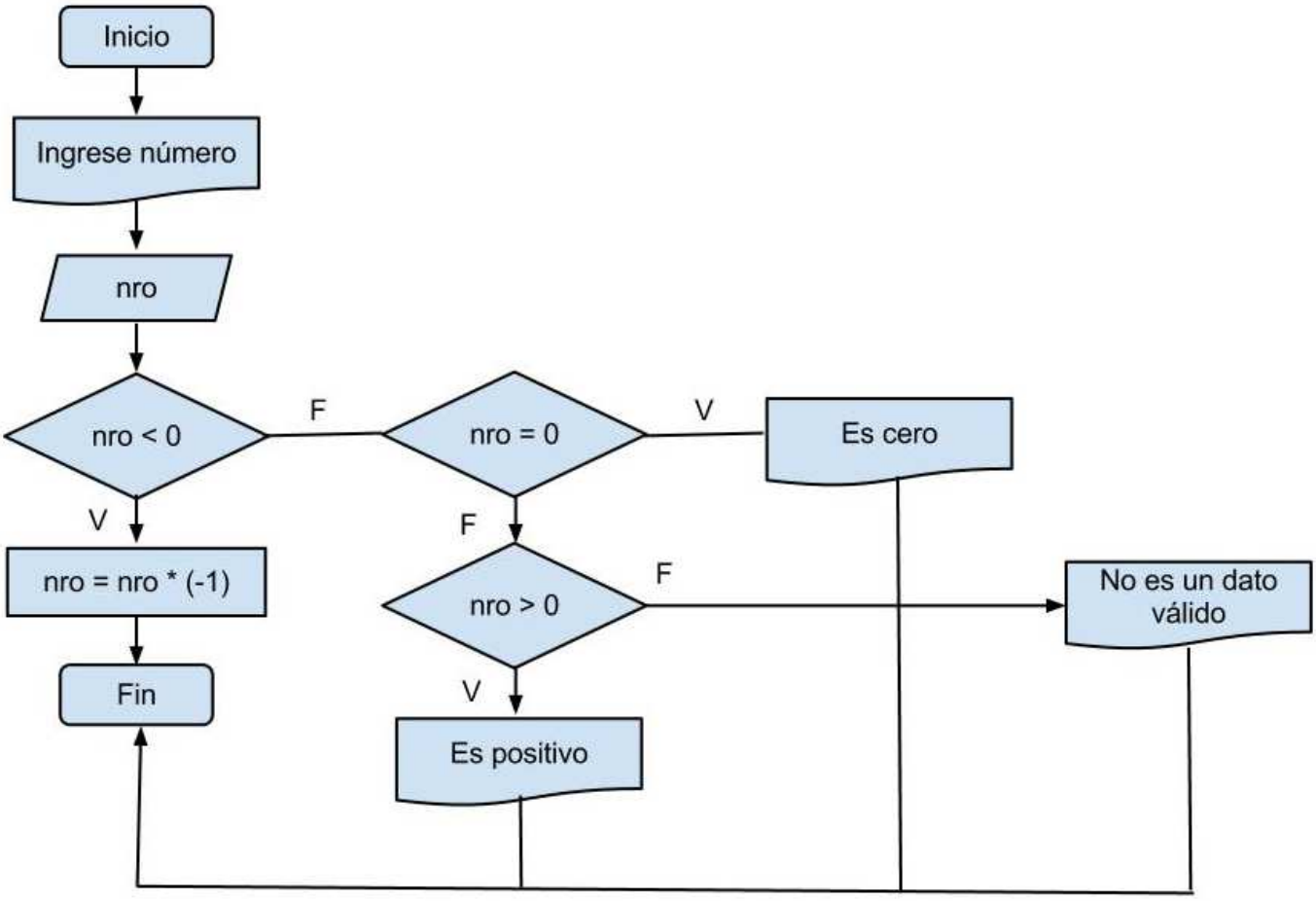

```
programa numeroPositivo

inicio
    mostrar: "Ingrese número"
    ingresar: nro
    si nro < 0 entonces
        nro = nro * (-1)
    sino si nro = 0 entonces
        mostrar: "Es cero"
    sino
        mostrar: "Es positivo"
    fin si
fin
```

En este caso, se agrega la **palabra reservada "sino si",** que significa que se vuelve a realizar otra condición o pregunta. En la mayoría de los lenguajes que implementan esta estructura, es posible agregar tantos "sino si" como sea necesario. Posteriormente veremos que una gran cantidad de condiciones puede ser reemplazada por otra estructura más adecuada (las condiciones múltiples), por lo que no es del todo recomendable utilizar este tipo de estructuras, pero no deja de ser importante conocerla.

3.4 Condición anidada

Estas estructuras deben utilizarse en casos puntuales, en los que sabemos que el algoritmo y su flujo tienen una cantidad finita y conocida de condiciones.

El funcionamiento es similar al de los ejemplos anteriores, manteniéndose el mismo conjunto de símbolos en el diagrama de flujo y las mismas palabras reservadas en el seudocódigo.



En este caso, se agregan una serie de condiciones a modo de ejemplo. Primero, como siempre, se verifica si el número es negativo y se lo transforma a positivo. Si no se cumple esta condición (continuando con el ejemplo anterior), se verifica si el número ingresado es 0 (cero), mostrándose un mensaje en caso verdadero. Si no es así, se lo vuelve a examinar , para saber si es un número positivo, y se informa si es así, solamente a fines informativos. En caso de no cumplirse tampoco con esta condición, podemos decir que lo que se ingresó no es un número o no es un dato correcto (no es negativo, ni cero, ni positivo, ergo, no un número que pueda ser interpretado por el programa). Cada lenguaje de programación tendrá su forma de realizar este tipo de validaciones, aunque tomamos esto como un caso ejemplificador, no representativo de las implementaciones de los lenguajes de programación.

```
programa numeroPositivo

inicio
    mostrar: "Ingrese número"
    ingresar: nro
    si nro < 0 entonces
        nro = nro * (-1)
    sino
        si nro = 0 entonces
            mostrar: "Es cero"
        sino
            si nro > 0 entonces
                mostrar: "Es positivo"
            sino
                mostrar: "No es un dato válido"
            fin si
        fin si
    fin si
fin
```

En este seudocódigo podemos comenzar a apreciar cómo la indentación va sumando participación e importancia: de esta forma se pueden visualizar claramente los "bloques" de código por la profundidad dada a cada nivel de la estructura. En forma ordenada y a primera vista se observa qué "si" se corresponde con su "fin si", por ejemplo. En este caso, donde tenemos tres preguntas una dentro de otra ("anidadas") es donde comienza a ser fundamental prestar atención a la legibilidad del código.

3.5 Condiciones múltiples

Este tipo de estructuras permite presentar una indeterminada cantidad de condiciones. Se verá cierta similitud con las dos estructuras anteriores, debido a que todas soportan múltiples preguntas. Aquí es donde entra en juego el criterio de programador (claramente acompañado de conocimientos técnicos) para seleccionar la estructura adecuada a cada tipo de condición.

Es común encontrarnos inicialmente con una condición que (a priori) podría resultar sencilla, pero al indagar más profundamente en los requerimientos (o debido a cambios en los mismos) estas situaciones nos llevan a ir creando estructuras complejas (muy de a poco), sobre las que recién terminamos de tomar conciencia cuando releemos el código escrito y ni siquiera nosotros somos capaces de entender lo que quisimos hacer. No hace falta aclarar qué cosa sucedería si este código llegara a manos de un colega quien debiera modificarlo o (simplemente) entenderlo.

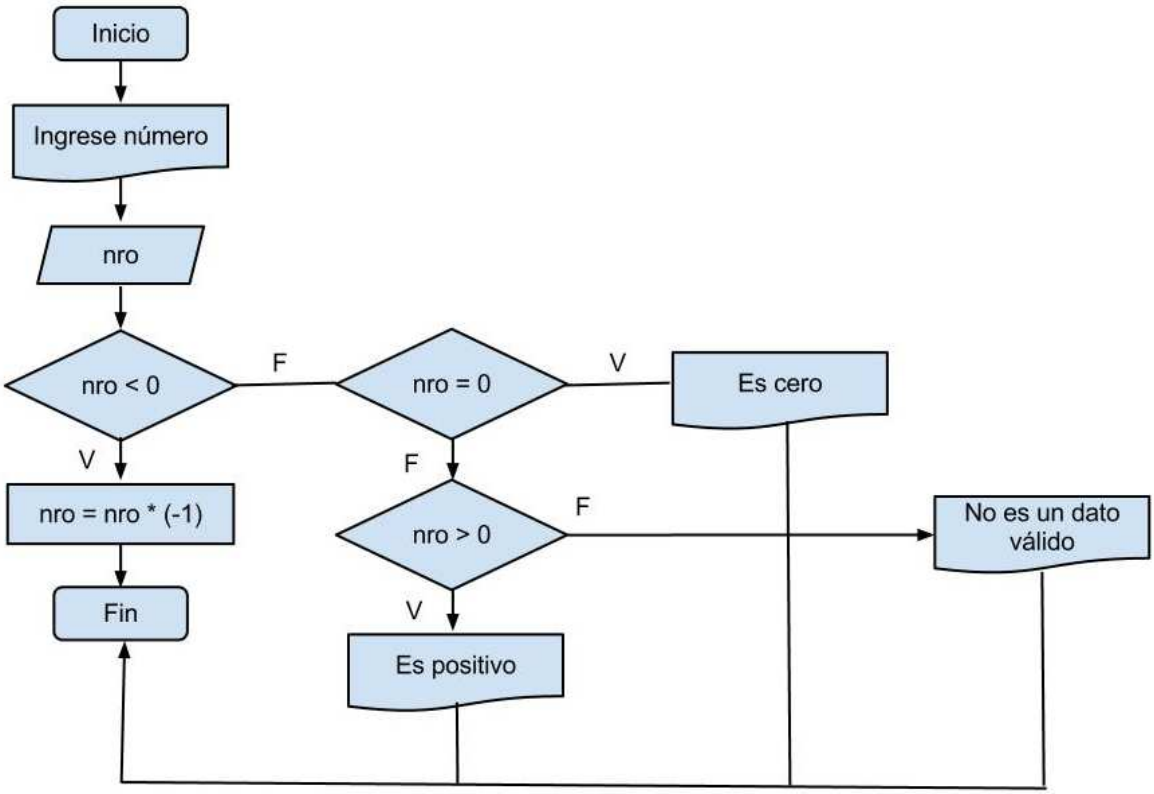
En este caso, esta estructura es siempre recomendada para casos en los que se cuenta con 3 o más condiciones a evaluar.

Éste es un ejemplo de algo que merece un curso completamente aparte, que nombraremos solamente para fijar el concepto: *refactoring*. Este término trae aparejado una disciplina que, como todas, tiene sus defensores y detractores. Para llevarla a un caso muy simple, podríamos decir que deberíamos hacer *refactoring* cuando descubrimos una pieza de código (propia o ajena, y siempre y cuando los tiempos y recursos del proyecto lo permitan) y ya sea por error, omisión o complejidad innecesaria, modificamos ese código con ánimos de corregirlo, mejorarlo y simplificarlo. Sin ahondar demasiado en este concepto, lo resumiremos como la actividad en la cual se toma un código existente y se modifica con un objetivo determinado.

Es importante siempre tener en cuenta el siguiente concepto, que también podemos decir que es una máxima:

"Somos los creadores de una pieza de software que sabemos cómo y dónde se inicia, pero no podemos saber cuántas otras personas lo tendrán que mantener en el futuro"

Volviendo a nuestro ejemplo, podemos ver que se agregan nuevas palabras reservadas en el seudocódigo, manteniéndose los mismos símbolos en el diagrama de flujo (de hecho, el diagrama es el mismo), aunque en algunos casos se utiliza para esta estructura un símbolo nuevo (en forma de octágono), que evitaremos usar para no agregar mayor complejidad a los diagramas.



```
programa numeroPositivo

inicio
    mostrar: "Ingrese número"
    ingresar: nro
    en caso de nro hacer
       caso es < 0
          nro = nro * (-1)
       fin caso
       caso 0
            mostrar: "Es cero"
        fin caso
        caso es > 0
            mostrar: "Es positivo"
        fin caso
        sino
            mostrar: "No es un dato válido"
    fin en caso de
fin
```

Podemos ver cómo la **estructura en-caso-de toma** de parámetro de comparación el valor ingresado (“nro”) y lo aplica a cada uno de los casos (en este ejemplo, son tres casos de comparación). En el primero y tercero, donde se hacen comparaciones con valores, se usa la estructura “caso es…”. En el segundo caso, que se compara directamente con un valor en particular se omite la cláusula “es”.

Si no se cumple con ninguno de los tres casos evaluados, ingresará en la opción de “sino”, que sería una especie de opción alternativa genérica si no se cumple ningún caso.

Las estructuras condicionales son la implementación concreta de las "validaciones" que deben realizarse en todos los programas.

¿Recuerdan el intento de encender el auto? (Unidad 3)

ACTIVIDAD 2:

Teniendo en cuenta los diferentes tipos de Estructuras Condicionales vistas: ¿Qué ejemplos se les ocurren de cada uno y porqué los elegirían?

- La respuesta debe hacerse en pseudocódigo.
- Elegir entre 1 y 3 estructuras condicionales, realizando 1 ejemplo de cada.
- Postear todos los ejemplos en un único tema nuevo por alumno en el foro.

Comparta sus respuestas en el foro de actividades de la Unidad.

4. Estructuras comunes

Existen otras estructuras de uso común, que ya vimos en la práctica, pero que vale la pena formalizar, ya que por ese uso difundido ya cuentan con nombre propio.

Vale aclarar que estas estructuras no son propias y exclusivas de programas estructurados, ya que no dependen de un paradigma en particular.

4.1 Asignaciones

Este tipo de instrucciones se refieren a la valoración de una variable a través de la asignación de un valor ingresado o del resultado de un proceso.

En este ejemplo, podemos ver cómo se asigna un valor fijo en “nro”, que posteriormente se muestra, junto con un mensaje. Así como en este caso se le asigna un “2”, también es posible, por ejemplo, hacer asignaciones del tipo "a = b", donde a "a" se le asigna el valor de "b" que, a priori, desconocemos.

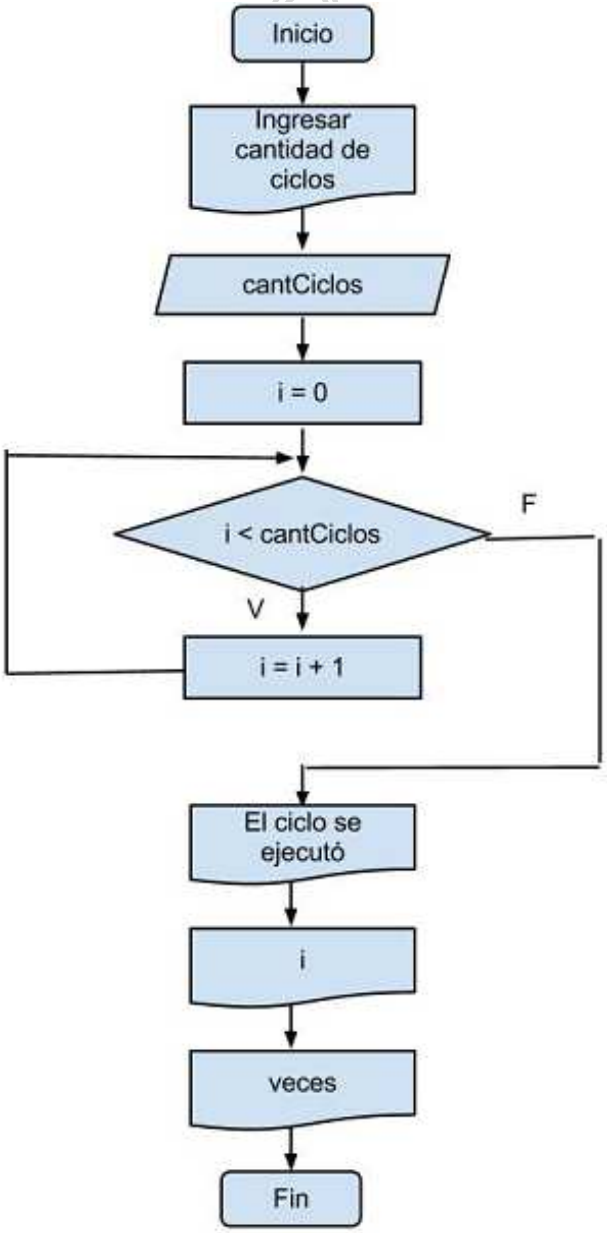
Este tipo de estructura no suele agregar mayor valor por sí mismo, pero sí al usarse junto a otras estructuras de mayor complejidad.


```
programa asignacion

inicio
    nro = 2
    mostrar: "El número es: "
    mostrar: nro
fin
```

4.2 Contadores

Los contadores son asignaciones incluidas en estructuras repetitivas, que a medida que el ciclo avanza, van incrementando un valor de en 1.

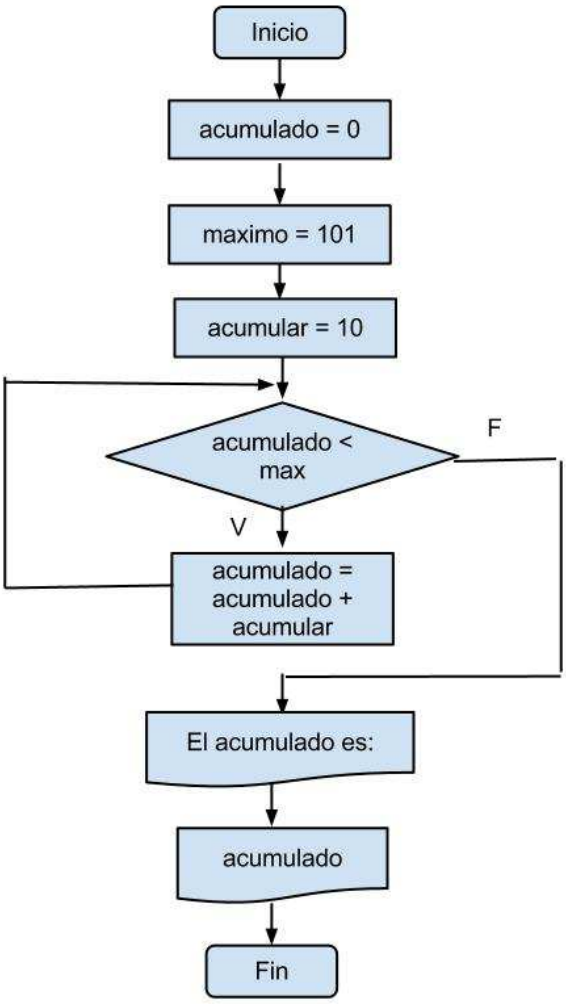
Como su nombre lo indica, permiten saber cuántas veces se ejecutó un ciclo repetitivo.




```
programa contador

inicio
    mostrar: "Ingresar cantidad de ciclos: "
    ingresar: cantidadDeCiclos
    i = 0
    mientras i < cantidadDeCiclos hacer
        i = i + 1
    fin mientras
    mostrar: "El ciclo se ejecutó "
    mostrar: i
    mostrar: " veces"
fin
```

4.3 Acumuladores

De mecánica similar a los contadores, los acumuladores cumplen una función que, como su nombre lo indica, condensan ("acumulan") en su valor actual un valor adicional, que puede ser fijo o variable.




```
programa acumulador

inicio
    acumulado = 0
    maximo = 101
    acumular = 10
    mientras acumulado < max hacer
        acumulado = acumulado + acumular
    fin mientras
    mostrar: "El acumulado es: "
    mostrar: acumulado
fin
```

Así como tenemos este ejemplo de acumulador cuya estructura es:

- **acumulado = acumulado + acumular**

Donde sabemos que "acumular = 10", un código equivalente, aunque no recomendable sería:

- **acumulado = acumulado + 10**

No sería recomendable del todo su uso, dada la similitud con la estructura Contador. La verdadera justificación del Acumulador es que el valor de "acumular" puede ir variando. Supongamos el caso visto anteriormente de las propina, cuyo monto va cambiando día a día. Si quisiéramos calculo

```
programa CuentaPropinas
   inicio
      propinasAcumuladas = 0
      cantidad = 1
      propina = 0

      mientras cantidad <= 7 hacer
         mostrar: "Ingrese propina del día"
         ingresar: propina
         propinasAcumuladas = propinasAcumuladas + propina
         cantidad = cantidad + 1
      fin mientras
fin
```

A

B

Analicemos este ejemplo:

En el bloque de código A, lo que tenemos son los valores iniciales que luego vamos a utilizar para hacer los cálculos. Siempre los definimos al inicio de programa y les asignamos (ver **Estructura Asignación**) valores iniciales.

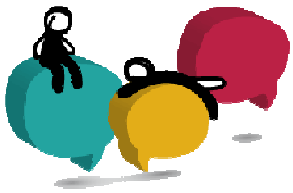
- "cantidad" va a ser la cantidad de días sobre los que se ingresarán propinas. Por ese motivo la **condición de salida** del ciclo mientras (ver **Estructura Repetitiva**) será menor o igual a 7, iniciándose en 1. Por lo tanto, este ciclo realizará 7 bucles: uno para cada día de la semana
- Sobre el resto de los valores no se tendrá referencia en esta instancia, por lo que los creamos con valor cero ("=0"), lo cual representa una buena práctica.

En el bloque B de código es donde sucede la magia:

- Sabemos que el ciclo mientras se va a repetir 7 veces
- Entonces, por cada repetición, nuestro programa pedirá que se ingrese el valor de la propina del día
- Este valor lo ACUMULAREMOS por cada repetición del ciclo. Esto significa que tomaremos el valor acumulado anterior y le sumaremos el nuevo valor. Esto lo vemos en la línea "**propinasAcumuladas = propinasAcumuladas + propina**".
    - La primera vez no se acumulará nada, dado que no hay valor anterior. Pero sabiendo que "algo + 0 = algo", podremos ver que la acumulación comienza a darse concretamente a partir de la segunda vuelta del ciclo.
- Luego de esto, tendremos un CONTADOR que nos permitirá "pasar de un día al otro" y será la estructura que hará que el ciclo, justamente, cicle. Esto lo podemos ver en la línea "**cantidad = cantidad + 1**".

De esta forma, podemos ver en este ejercicio integrador algunos de los temas vistos en esta Unidad y en la anterior, combinados:

- **Asignaciones**
- **Operaciones lógicas**
- **Operaciones Aritméticas**
- **Contadores**
- **Acumuladores**
- **Estructuras repetitivas**
- **Estructuras condicionales (como complemento de las repetitivas)**
- **Y, como todo algoritmo, una secuencialidad**



Actividades de la Unidad 5

- Punto 1: Hacer un programa en diagrama de flujo que muestre los números del 100 al 0, en orden decreciente.

- Punto 2: Escribir un programa en pseudocódigo que pida el ingreso de una letra y que valide si esa letra es una vocal. Mostrar un mensaje tanto si es vocal como si no lo es.

Los ejercicios deberán estar en un documento Microsoft Word compatible y deberán realizar la entrega en un link publicado en el Campus para tal fin.

- **¡Atención! Esta actividad no se debe subir a los foros públicos, sino deberá ser entregada en el lugar especialmente identificado para este caso.**

- **¡Atención! En caso de no hacer la entrega a tiempo, como se estipula en los mensajes y foros del Campus virtual, NO se debe realizar la entrega en los foros públicos. Las fechas y plazos son los mismos para todos y estos deberán respetarse.**

Lo que vimos

- Principales aspectos del paradigma estructurado de programación
- Estructuras de control y sus principales variantes
- Otras estructuras comunes
- Ejemplo integrador

Lo que viene:

- Uso y funcionalidad de las variables
- Uso y funcionalidad de las constantes
- Uso de tipos de datos para variables y constantes
- Conceptos y uso de los vectores y matrices
- Uso de funciones y conocer los tipos