UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

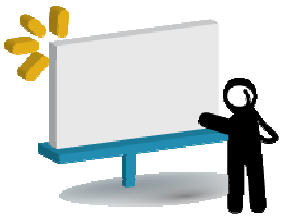
FUNDAMENTOS DE LA PROGRAMACIÓN

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 pseudocódigo.

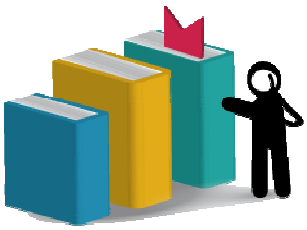
También tendremos la oportunidad de analizar las estructuras de control principales: la estructura secuencia, la estructura condicionales y la estructura repetitivas y las variantes de cada una.

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

Objetivos:

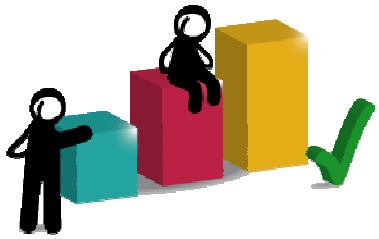
Que los participantes:

- Comprendan los principales aspectos del paradigma estructurado de programación
- Conozcan el uso funcionalidad de las variables
- Comprendan el uso y funcionalidad de las constantes
- Conozcan el uso de tipos de datos para variables y constantes
- 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. Variables
    2.1 Definición
    2.2 Convenciones de nomenclatura
    2.3 Constantes
3. Tipos de datos
    3.1 Definición
    3.2 Tipos
    3.3 Declaración de variables
4. Detalles de estructuras básicas de control
    4.1 Estructura secuencial
    4.2 Estructura condicional o selectiva
    4.3 Estructura repetitiva o iterativa
5. Tipos de estructuras condicionales
    5.1 Condición simple



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:

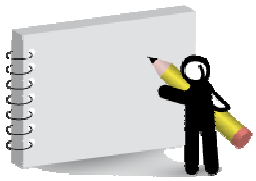
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

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:

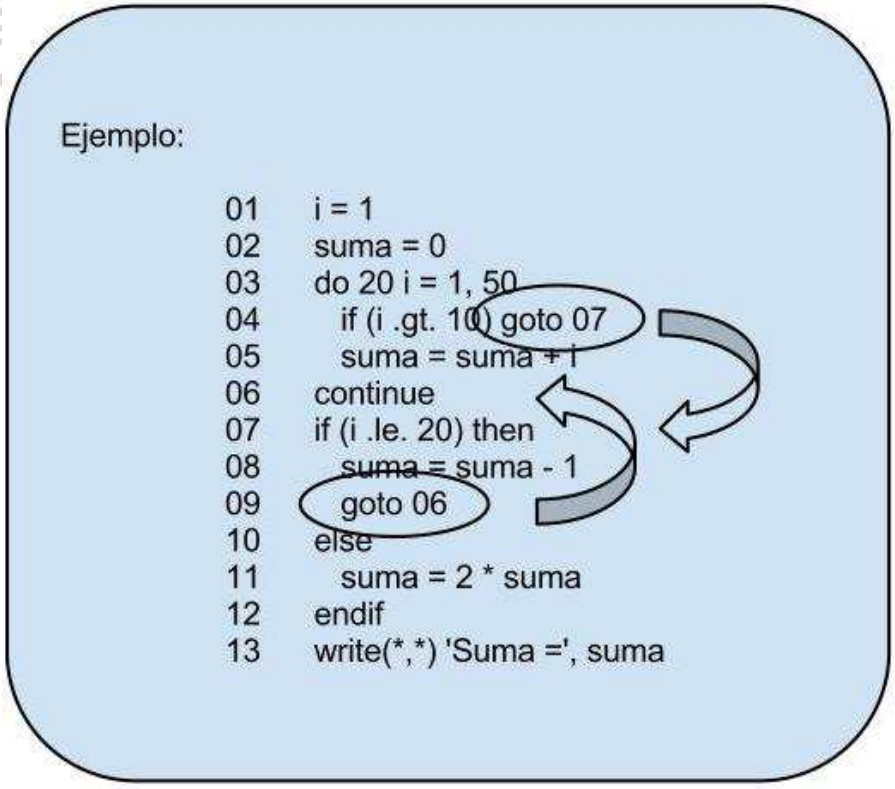
- **Secuencia**: tal como lo venimos viendo, una instrucción se ejecuta una detrás de otra, siendo este orden conocido

- **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

2. Variables

2.1 Definición

Si bien las venimos nombrando y utilizando informalmente, es necesario adentrarnos algo más en el concepto de “variable”, dado que su uso y convenciones así lo requieren.

Si metaforizáramos sobre el uso de las variables, diríamos que son distintos **recipientes donde colocamos distintos tipos de elementos**. Es eso lo que venimos haciendo hasta este momento cuando decimos que asignamos un valor a una variable, por ejemplo:

```
miNumero = 1555551234
nombreDeMiMascota = "Lis"
```

Pero no todos los recipientes son iguales, eso está claro. Imaginemos que tenemos diferente recipientes, por ejemplo:

- un tubo de ensayo
- una bolsa
- una canasta

Por otro lado, tenemos que llenarlos con estos contenidos:

- un puñado de clavos
- un líquido
- pan

Si bien físicamente sería posible colorar el pan en el tubo de ensayo, el líquido en la bolsa y los clavos en la canasta, nos deberíamos preguntar: ¿Es lo mismo cualquier elemento en cualquier recipiente? ¿Conceptualmente, tiene sentido?

Lo que está claro es que si existe un tipo definido de recipiente para un tipo definido de contenido, el sentido común nos lleva a vincular uno con otro, según corresponda. Lo mismo ocurre con las variables: sus "tipos" (que diferenciaremos más adelante en esta misma Unidad) son heterogéneos y su uso debe estar ligado a su razón de ser. **No es lo mismo (más allá que algún lenguaje lo permita, siendo esto conceptualmente dudoso) asignar un número a un tipo de variable preparado para contener caracteres.**

Para esto, vamos a realizar lo que se conoce como **declaraciones de variables**. Esto significa que explícitamente vamos a crear una variable diciendo qué tipo de datos va a poder contener. De esta forma, cuando programemos una rutina en un lenguaje en particular, le estarán diciendo al compilador (ver tema Compiladores del Módulo 1) para qué vamos a utilizar esa variable. Y el compilador será capaz de advertirnos si ese uso es correcto o no, ya que va a verificar si realmente le estamos asignando los valores del tipo que espera recibir.

2.2 Convenciones de Nomenclatura

Por regla general, todos los lenguajes de programación sugieren realizar las declaraciones y asignaciones en la parte superior e inicial del programa. Esto aporta limpieza y claridad al código, permitiendo ver de un vistazo cuáles son las variables que serán utilizadas en un programa, aunque posteriormente éstas serán manipuladas (modificadas, vaciadas, consultadas…) en otro punto del programa.

Sobre las variables anteriormente dijimos que no existe un estándar ni un único criterio unificado para nombrarlas. Aunque en este curso establecimos como regla general el uso del método llamado lowerCamelCase, el cual respetaremos de ahora en adelante.

Por otro lado, en los lenguajes de programación modernos encontramos (como regla genérica, pero no absoluta) que las asignaciones de valores numéricos se realizan igualando una variable con el valor (por eso **miNumero = 1555551234**), a diferencia de un valor alfanumérico (letras, números, caracteres especiales), cuya asignación se realiza igualando la variable al valor entre comillas (por eso **nombreDeMiMascota = "Lis"**).

Para el tratamiento de fechas, cada lenguaje suele implementar su forma propia de asignación, aunque a fines prácticos en este curso lo utilizaremos con la misma convención que usamos con los caracteres: el valor va a ir entre comillas dobles (""). Siempre evitaremos incluir en el nombre de las variables caracteres especiales (que no sean las letras del abecedario inglés). **Esto incluye las letras "ñ" y (especialmente) las tildes en las letras acentuadas.** Más allá de que algunos compiladores soporten estos caracteres, no resulta una buena práctica de programación. Es muy común encontrar variables como "año" o "número", por lo que deben ser cuidadosamente descartadas. Una posible variante de "año" podría ser, por ejemplo "anio", para mantener una pronunciación y comprensión adecuada del significado de la variable.

Finalmente, una recomendación importante: las variables deben tener nombres significativos. Es muy importante evitar el uso de variables del tipo "var1" o "pepe", por más extraño que parezca. Las variables siempre (SIEMPRE) deben tener un nombre que se corresponda con lo que vayan a contener. Si tenemos fechas y no sabemos a qué fechas nos estaremos refiriendo, simplemente tendremos una variable llamada "fecha". Si tenemos un rango de fechas, tendremos "fechaInicial" y "fechaFinal". Si tenemos una fecha de cumpleaños, será "fechaDeNacimiento", etc. En este caso, entra en juego una de las principales habilidades del programador: **el sentido común**.

2.3 Constantes

En contraposición (incluso de conceptos) con respecto a las variables, encontramos ciertos tipos llamado Constantes, dado que se les suele asignar un valor en forma inicial (cuando se declara o en forma posterior) y su valor no suele cambiar a lo largo del programa.

Este tipo de valores estáticos resultan de gran utilidad (por su simpleza y por la claridad que le aportan al código) a la hora de realizar comparaciones y/o ante la necesidad de tener listas de valores que sabemos que no van a cambiar. Por ejemplo: los días de la semana, los meses, marcas de autos, etc.

Como regla general, también se declaran y se les asignan valores al principio de un programa.

Para nombrarlas, siendo concordantes con la mayoría de los lenguajes de programación modernos, utilizaremos MAYUSCULAS, a diferencia del lowerCamelCase definido anteriormente para las variables.

Cabe destacar que en realidad sí es posible cambiar durante la ejecución de un programa (en algunos lenguajes) el valor de una constante, salvo que ésta haya sido declarada de una forma especial, evitando las posibilidades de cambios, aunque modificar el valor representaría una mala práctica de programación. En el caso en que sospechemos que una constante pueda tomar un valor distinto al declarado inicialmente, siempre deberemos declararla como variable y no como constante (regla general: ante la duda, siempre es variable).

3. Tipos de datos

3.1 Definición

Los tipos de datos, más allá de las extrañas metáforas de clavos y tubos de ensayo, son uno de los componentes más importantes y sensibles de los lenguajes de programación.

Decimos que son importantes porque, a través de la combinación de pequeñas variables, cada una de su tipo, logramos representar en forma abstracta una realidad compleja.

Aunque también son sensibles, ya que es uno de los puntos más susceptibles a fallas y errores por parte del programador. Para explicar esto, antes daremos algunas definiciones breves sobre términos comúnmente usados:

- **Tiempo de programación**: se refiere al momento y las actividades realizadas por el programador durante la etapa de construcción de un programa.

- **Tiempo de compilación**: hace referencia a todas aquellas actividades y pasos que realiza internamente un compilador, cuando un programador ordena la compilación de un programa que se encuentra construyendo.

- **Tiempo de ejecución**: se refiere al uso que le da un usuario a un programa previamente desarrollado y compilado por un programador. Por "usuario" entendemos a cualquier persona que ejecute el programa, pudiendo ser el mismo programador, en caso de estar realizando pruebas, por ejemplo.

Volviendo a la "sensibilidad" de los tipos de datos de las variables, es extremadamente común que una variable que concebimos en tiempo de programación pensando que iba a recibir un número, en tiempo de ejecución finalmente puede terminar recibiendo una letra, con la consecuente falla en el programa, al tratarse de dos tipos de datos incompatibles.

¿Por qué ocurriría esto? En nuestro ejemplo anterior de los días de la semana: ¿Qué ocurriría si el usuario en lugar de ingresar un número del 1 al 7, ingresar los siguiente: "/(uy687f"? La respuesta es simple: al tratar de asignar ese valor a la variable "dia", se produciría una falla en el programa. Una falla que al no estar preparados para capturarla, ocasionaría una interrupción en la ejecución del programa.

Esto es con lo que tenemos que lidiar: el tiempo de programación es un escenario estático, en el que controlamos todo lo que ocurre. Por el contrario, el tiempo de ejecución es un escenario dinámico, donde las posibilidades de multiplican más allá de las variantes que podemos imaginar o prever.

La primera reacción podría ser: "¿Cómo es posible que un usuario sea tan ***** para ingresar "/(uy687f" cuando le estamos pidiendo explícitamente que "Ingrese número de día de la semana"? … La respuesta es que lamentablemente tendremos que lidiar con estos factores externos que condicionan nuestro trabajo de programación. Sean usuarios, sean tiempos o costos, sean máquinas de baja calidad, siempre deberemos tener en cuenta los aspectos contextuales a la hora de encarar un desarrollo de software.

Finalizando, para darle un cierre al marco teórico de los tipos de datos, podemos decir que se trata de definir qué clase de información puede contener una variable.

3.2 Tipos

Dependiendo de la implementación de cada lenguaje de programación en particular, podemos encontrar distintos tipos de datos en cuanto a nombres (tipos de datos "string" y "str", que tienen la misma función) y capacidades (cantidad de números detrás de la coma que soportan dos tipos de datos similares pero de diferentes lenguajes de programación).

La siguiente es una tabla comparativa de los tipos de datos más usados:

Tipo de dato	Tamaño	Descripción
Boolean	1 byte	Valores Verdadero/Falso.
Date	8 bytes	Una fecha entre los años 100 y 9999
Float	4 bytes	Un valor de coma flotante entre – 3,402823E38 y – 1,401298E-45 para valores negativos, y desde 1,401298E-45 a 3,402823E38 para valores positivos, y 0.
Double	8 bytes	Un valor de coma flotante entre – 1,79769313486232E308 y – 4,94065645841247E-324 para valores negativos, y desde 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos, y 0.
Integer	4 bytes	Un entero entre – 2.147.483.648 y 2.147.483.647.
String	2 bytes por carácter	Desde 0 a 255 caracteres.

A fines prácticos, definiremos las siguientes **palabras reservadas** como un subconjunto de tipos de datos válidos para este curso:

- **string** (“cadena” por “**cadena** de caracteres”): lo usaremos para identificar variables que contienen de 0 a 255 caracteres. Contiene valores alfanuméricos (letras, números, símbolos de puntuación, etc.)
- **integer** (“entero” de “número entero”): lo usaremos para representar números enteros (no decimales)
- **float** (“flotante” de “coma flotante”): lo usaremos para representar números con coma (decimales)
- **date** (“fecha”): lo usaremos para representar fechas
- **boolean** (“booleano” o lógico): lo usaremos para almacenar valores binarios (verdadero o falso), como por ejemplo, el resultado de una condición

Serán una de las pocas palabras en idioma inglés que utilizaremos en nuestro pseudocódigo, para evitar confusiones.

3.3 Declaración de variables

Como se dijo anteriormente, realizaremos la declaración de variables y su inicialización (primer valor que se asigna a una variable) al inicio del programa.

Un ejemplo podría ser:

```
programa DeclaracionDeVariables
inicio
        var date fechaDeNacimiento = "9/12/2018"
        var string nombre
        nombre = "JuanFer"
        var integer edad = 33
        var flot altura = 1,77
        var boolean tieneHijos
        tieneHijos = verdadero
fin
```

Anteponemos al tipo de dato la **palabra reservada "var"** para diferenciarlas de las constantes, a las cuales identificaremos con la palabra reservada "**const**". Posteriormente definiremos el tipo de dato, utilizando alguno de los cinco tipos que precisamos hasta ahora en el presente curso**: date, string, integer, float y boolean**.

Vemos también cómo se puede realizar de dos formas distintas las inicializaciones de las variables:

- Junto con la declaración, como es el caso de fechaDeNacimiento, edad y altura
- Luego de la declaración, como en el caso de nombre o tieneHijos

Finalmente, prestemos atención a la forma en que se realizan las inicializaciones:

- Valores de date y string se van a asignar a las variables utilizando las comillas dobles (“”)
- Valores de float e integer se van a asignar a las variables igualándolas con los números dados, lo mismo que los valores posibles de las variables booleanas (verdadero y falso, sin comillas)

Resumiendo:

- **La variables sirven para almacenar datos en forma temporal**
- **Existen diferentes TIPOS de variables relacionadas con las características de los datos que van a contener**
- **Las constantes sirven para almacenar datos que no van a ser modificados**
- **Tanto variables como constantes deberían definirse al inicio del programa**
- **Tanto variables como constantes deben tener un tipo y los valores que se le asignen deben ser del mismo tipo que fuera declarada la variable o constante**

IMPORTANTE:

De ahora en adelante, TODAS las variables y constantes que vayan a utilizar deberán ser DECLARADAS en forma previa a su uso:

- **No se puede ingresar o asignar un valor en una variable si no se declaró la variable**
- **No se puede comparar un valor contra una constante si no se declaró la constante**

4. Detalles de estructuras básicas de control

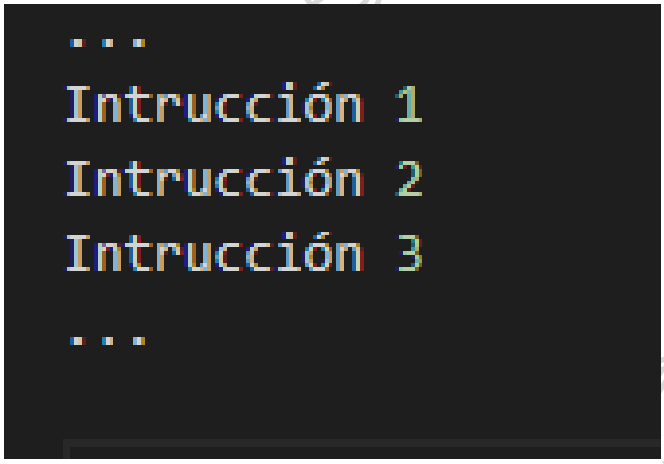
Como se dijo anteriormente, la programación estructurada tiene como principales características:

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

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

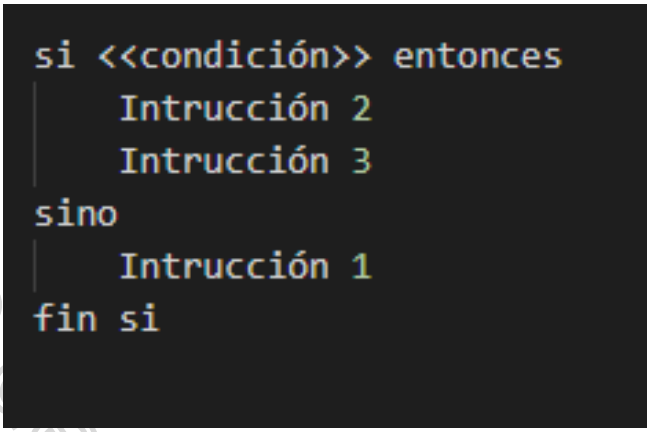
4.1 Estructura secuencial

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.



4.2 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).



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”.

En el caso de que la <<condición>> sea falsa, 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
- nombre = "Carlos"
- mensaje != "OK"

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

4.3 Estructura repetitiva o iterativa

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

```
Intrucción 1
mientras <<condición>> hacer
    Intrucción 1
fin mientras
Intrucción 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**.

5. 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.

5.1 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
    var integer nro
    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.

5.2 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 pseudocódigo, que indica el flujo alternativo que se ejecuta en caso de no cumplirse la condición dada.

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

5.3 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.

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.

```
programa NumeroPositivo
inicio
    var integer nro
    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
```

5.4 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 palabras reservadas en el pseudocódigo.

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
    var integer nro
    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 pseudocódigo 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.

5.5 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 tener una complejidad alta 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 pseudocódigo.

```
programa NumeroPositivo
inicio
    var integer nro
    mostrar: "Ingrese número"
    ingresar: nro
    en caso de nro hacer
        caso < 0
            nro = nro * (-1)
        fin caso
        caso = 0
            mostrar: "Es cero"
        fin caso
        caso > 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 …”. 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 DE ANÁLISIS - ESTRUCTURAS:

Teniendo en cuenta los diferentes tipos de Estructuras Condicionales resolver los siguientes requerimientos:

Estructuras condicionales simples:

1) Crear un programa que solicite el ingreso de la temperatura corporal y muestre un mensaje en caso de tener fiebre

2) Crear un programa que pida la cantidad la cantidad de puertas que tiene un vehículo y valida si es coupé y muestre un mensaje en forma afirmativa

3) Crear un programa que solicite el ingreso de una edad e indique si puede votar (a partir de los 16 años)

4) Crear un programa que solicite el ingreso de la altura de una persona e informe si puede acceder a un juego mecánico (el mínimo para ingresar es 1,20 metros)

Estructuras condicionales dobles:

5) Crear un programa que pida el ingreso de una clave, valide contra una clave guardada si coincide o no y que lo informe, sea caso afirmativo o negativo

6) Crear un programa que pida el ingreso de la cantidad de puertas de un auto, las valide e informe si se trata o no de un auto tipo sedán

7) Crear un programa que valide si los últimos 3 números de la patente es un número par o no. Si es par, indicar que puede circular. Si no es par, indicar que no puede circular.

8) Crear un programa que pida el ingreso de la edad de una persona y que indique si es o no es mayor de edad

Estructuras condicionales anidadas:

9) Crear un programa que, dada la altura de una persona ingresada por el usuario, indique si se trata de una persona alta (más de 1,7 metros), una persona de altura media (entre 1,50 metros y 1,70 metros) o una persona de altura baja (menos de 1,5 metros).

10) Crear un programa que pida el ingreso de un año y que valide si se encuentra dentro entre el año 2000 y el año 2010. Informar caso afirmativo o casos negativos.

11) Crear un programa que, a partir del ingreso de la edad de una persona informe si está obligada a votar (más de 18 años), si no está obligada (más de 70 años) o si no puede votar.

12) Crear un programa que, dado el ingreso por parte del usuario de la posición de un auto en salida de una carrera, indique si el auto está en la primera fila, si es el primero, si está en la segunda fila, si se encuentra en el medio (hasta la 8va fila incluida) o si se encuentra al final.

Estructuras condicionales compuestas:

13) Crear un programa que, a partir del ingreso del valor de una compra indique si el envío por flete es gratis (más de $2000), si está bonificado (valor entre $1000 y $1999, y envío de $150) o si se paga envío completo ($250)

14) Crear un programa que pida el ingreso de la forma de pago y que la valide: si es en efectivo, tomarlo. Si es con tarjeta, rechazarlo en forma momentánea. Si es otro medio de pago, indicar que no es válida.

15) Crear un programa que pida el ingreso del nombre de la radio favorita del usuario y que indique si la respuesta es correcta (si es que se trata de radio "Horizonte 94.3" o lo contrario).

16) Crear un programa que pida el ingreso de una nota cuantitativa de un examen y que muestre la nota cualitativa (si es 10 o 9: muy sobresaliente, si es entre 6 y 8: sobresaliente, si es menos de 6: desaprobado). En caso de no tratarse de ninguno de estos valores indicar que la nota cuantitativa (numérica) no es válida.

Estructuras condicionales múltiples:

17) Crear un programa que, a partir del ingreso del código telefónico de área, indique si es de Santa Fe (0341), si es de Entre Ríos (0343), si es de Buenos Aires (011) o si es un código de área desconocido.

18) Crear un programa que permita obtener las notas de un examen multiplechoise a partir de la cantidad de aciertos, siendo: entre 100 y 90: sobresaliente, entre 89 y 70: notable, entre 69 y 50: bien, entre 49 y 40: aprobado, y menos de 40: suspenso).

19) Crear un programa que indique la remuneración anual (12 sueldos + 2 medio aguinaldos, que se obtienen de 6 sueldos sobre los 12 meses) y el bono por presentismo (que se obtiene del sueldo mensual divido cuatro) para UNA de las siguientes posiciones: gerente (el sueldo es de $100.000), el supervisor (sueldo de $55.000) o asistente ($23.000).

20) Crear un programa que indique en qué estación del años estamos, partir del ingreso de la primera letra de la estación correspondiente.

Podrán comparar sus respuestas con los ejemplos resueltos en el "ANEXO 1 - Respuesta Actividad de Análisis - Estructuras" al final de esta unidad.

Previamente pueden consultar el "ANEXO 3 - Checklist de código".

IMPORTANTE:

- **¡Traten de resolverlo cada uno por sus propios medios!**
- **Las respuestas (el programa) NO debe ser enviado a los foros del Campus, salvo dudas puntuales de las respuestas.**

6. 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.

6.1 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
    var integer nro
    nro = 2
    mostrar: "El número es: "
    mostrar: nro
fin
```

6.2 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
    var integer i
    var integer cantidadDeCiclos

    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
```

6.3 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
    var integer acumulado
    var integer maximo
    var integer acumular

    acumulado = 0
    maximo = 101
    acumular = 10
    mientras acumulado < maximo 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
    var integer propinasAcumuladas
    var integer cantidad
    var integer propina

    //comienzo de Bloque A de código
    propinasAcumuladas = 0
    cantidad = 1
    propina = 0
    //fin de Bloque A de código

    //comienzo de Bloque B de código
    mientras cantidad <= 7 hacer
        mostrar: "Ingrese propina del día"
        ingresar: propina
        propinasAcumuladas = propinasAcumuladas + propina
        cantidad = cantidad + 1
    fin mientras
    //fin de Bloque B de código
fin
```


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**

7. Tips para el uso de condicionales y repeticiones

- Un "si" **siempre** lleva, como máximo un "fin si".
- No existe el "fin sino". El "sino" no se cierra, porque es parte del "si" que se cierra con el "fin si".
- El "en caso de" debe utilizarse cuando se van a evaluar más de 3 o 4 valores sobre una misma variable.
    - El "caso" puede ir únicamente dentro de un "en caso de".
    - Todo "caso" lleva su "fin caso".
    - La variable nunca va en el "caso", solo se menciona en el "en caso de".
    - No debería evaluarse más de un valor en cada "caso", ya que se pierde el sentido de prolijidad que aporta la estructura condicional múltiple "en caso de".
- Las estructuras condicionales anidadas deben tender a no ser utilizadas (terminan generando un código complejo y poco claro).
- Es posible usar múltiples condiciones en una estructura condicional. Esto se hace evaluando la variable contra cada valor esperado y uniendo cada condición con un operador lógico "Y" u "O".

Por ejemplo:

```
si valor > 22 O valor <  20 entonces
    //entra acá si valor es 23 o más grande o si
    //valor es 19 o más chico, nunca si valor es 20, 21 o 22.
fin si
```

Otro ejemplo:

```
si valor < 22 Y valor > 20 entonces
    //entra acá si valor es 21 o más grande o si valor es 21 o
    // más chico, o sea, es lo mismo que preguntar "valor = 21"
fin si
```

No hay límites en cuanto a la cantidad de condiciones, PERO: si van a evaluar una variable contra 3 o más valores, ya se vuelve conveniente usar el "en caso de", porque es más útil y claro para estos casos.

También es posible hacer combinaciones con otras variables.

Ejemplo:

```
nombre = "cosme"
clave = "fulanito"
intentos = 0

ingresar: nombreUsuario
ingresar: claveUsuario
intentos = intentos + 1

si nombre = nombreUsuario Y clave = claveUsuario Y intentos < 3 entonces
    mostrar: "acceso concedido"      //entra únicamente si se cumplen las 3 condiciones
sino
    mostrar: "acceso denegado"
fin si
```

También es posible combinar los conectores "Y" u "O".

```
si (nombre != nombreUsuario O clave != claveUsuario) Y intentos = 4 entonces
    mostrar: "clave bloqueada"      //entra si no se cumple alguna de las  2 primeras condiciones
                                    // y si se cumple la tercera ("intentos = 4")
fin si
```

En estos casos es conveniente valerse de paréntesis para dividir correctamente los términos de la condición compuesta.

- No tiene sentido hacer una comparación contra una variable que no tiene valor asignado o ingresado.

Ejemplo incorrecto:

```
programa EsGoleadaONo
inicio
    var integer goles

    si goles = 6 entonces
        mostrar: "Es goleada"
    sino
        mostrar: "No es goleada"
    fin si
fin
```

La variable "goles" no tiene valor asignado o ingresado, por lo que el resultado de la condición será siempre FALSO.

Ejemplo correcto:

```
programa ValidaIngreso
inicio
    var string clave
    var string intento

    clave = "2fdjifj89234"

    ingresar: intento

    si clave != intento entonces
        mostrar: "Clave incorrecta"
    sino
        mostrar: "Acceso permitido"
    fin si
fin
```

La variable "clave" tiene un valor asignado y la variable "intento" tiene un valor ingresado, por lo que es posible hacer la comparación.

- Si tenemos que realizar en forma sucesiva una serie de instrucciones iguales o muy similares, digamos, más de 4 o 5 veces, deberíamos contemplar el uso de una estructura repetitiva "mientras".

- Si una variable es parte de la condición de salida de un "mientras", y esa variable NO es modificada DENTRO del ciclo "mientras", es posible que se trate de un "loop infinito", o sea, el ciclo que no finaliza "nunca". Ejemplos:

```
//loop infinito
i = 0
mientras i < 100 hacer
    mostrar: "El valor de i es:"
    mostrar: i
fin mientras

//loop infinito
ingresar: pass
mientras pass != "GloriaEterna" hacer
    mostrar: "La password es incorrecta"
    mostrar: "Reintente"
fin mientras

//modo correcto
i = 0
mientras i < 100 hacer
    mostrar: "El valor de i es:"
    mostrar: i
    i = i + 1
fin mientras

//modo correcto
ingresar: pass
mientras pass != "GloriaEterna" hacer
    mostrar: "La password es incorrecta"
    mostrar: "Reintente"
    ingresar: pass
fin mientras
```

8. Ejercicio resuelto

```
programa DiaDeLaSemana
inicio
    var integer dia
    const integer LUNES = 1
    const integer MARTES = 2
    const integer MIERCOLES = 3
    const integer JUEVES = 4
    const integer VIERNES = 5
    const integer SABADO = 6
    const integer DOMINGO = 7
    const string ERROR = "El valor ingresado es incorrecto"

    mostrar: "Ingrese número de día de la semana: "
    ingresa: dia
    en caso de dia hacer
        caso LUNES
            mostrar: "Es LUNES"
        fin caso
        caso MARTES
            mostrar: "Es MARTES"
        fin caso
        caso MIERCOLES
            mostrar: "Es MIERCOLES"
        fin caso
        caso JUEVES
            mostrar: "Es JUEVES"
        fin caso
        caso VIERNES
            mostrar: "Es VIERNES"
        fin caso
        caso SABADO
            mostrar: "Es SABADO"
        fin caso
        caso DOMINGO
            mostrar: "Es DOMINGO"
        fin caso
        sino
            mostrar: ERROR
    fin en caso de
fin
```

En este simple ejemplo, vemos cómo inicialmente se declaran 7 constantes, cada una llamada como un día de la semana. A éstos, se les asigna un valor del 1 al 7 para identificarlos. Adicionalmente se declara otra constante llamada ERROR, a la cual se le asigna un mensaje.

Posteriormente se declara una estructura condicional “en caso de”. Cada uno de los “caso” compara el valor ingresado y guardado en la variable “dia”. En el caso de que el usuario haya ingresado un número entre 1 y 7, éste será detectado por la condición y se mostrará el mensaje que corresponde.

Hacer "caso LUNES", sería lo mismo que hacer "caso 1"; "caso MARTES" sería "caso 2" y así sucesivamente. Pero supongamos un extenso programa en el que cual se usan los días de la semana, digamos, 20 veces. Y un buen día nos llega el pedido para adaptarnos a una nueva regulación por la cual el día "1" de la semana pasa de Lunes a Domingo. Si usáramos los valores (1, 2, 3...) en forma directa, tendríamos que hacer el cambio en 20 lugares diferentes. Si en lugar de usar los valores, usamos CONSTANTES, sólo habría que cambiar la declaración de las constantes, sin tener que modificar el resto del programa.

Si el número ingresado no se encuentra en el rango antes mencionado, se muestra el mensaje de error guardado en la constante ERROR.

ACTIVIDAD DE ANÁLISIS - INTEGRACIÓN:

- Punto 1: Escribir un programa en pseudocódigo que muestre los números del 100 al 0, en orden decreciente. No deberá tener más de 20 líneas de código sin contar espacios en blanco y comentarios.
- 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.

Podrán comparar sus respuestas con los ejemplos resueltos en el "ANEXO 2 - Respuesta Actividad de Análisis - Integración" al final de esta unidad.

Previamente pueden consultar el "ANEXO 3 - Checklist de código".

IMPORTANTE:

- **¡Traten de resolverlo cada uno por sus propios medios!**
- **Las respuestas (el programa) NO debe ser enviado a los foros del Campus, salvo dudas puntuales de las respuestas.**

ANEXO 1 - Respuesta Actividad de Análisis - Estructuras

¿Trataron de resolver los requerimientos?

En caso afirmativo, avancen 3 páginas.

En caso negativo, ¡no avancen y traten de resolverlos antes de ver las respuestas!

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

//ESTRUCTURAS CONDICIONALES SIMPLES

1)

```
programa DeteccionDeFiebre
inicio
    var float temp
    mostrar: "Ingrese su temperatura"
    ingresar: temp
    si temp >= 37,5 entonces
        mostrar: "Usted tiene fiebre, contáctese con su médico"
    fin si
fin
```

2)

```
programa CantidadDePuertas
inicio
    var integer nroDePuertas
    mostrar: "Ingrese cantidad de puertas"
    ingresar: nroDePuertas

    si nroDepuertas = 2 entonces
        mostrar: "Su auto es coupe"
    fin si
fin
```

3)

```
programa EdadParaVotar
inicio
    var integer edad
    mostrar: "ingrese edad"
    ingresar: edad

    si edad >= 16 entonces
        mostrar: "Puede votar"
    fin si
fin
```

4)

```
programa AccesoPorAltura
inicio
    var float altura
    mostrar: "Ingrese altura"
    ingresar: altura

    si altura < 1,20 entonces
        mostrar: "No puede acceder a la atracción"
    fin si
fin
```

//ESTRUCTURAS CONDICIONALES DOBLES

5)

```
programa ClaveHomeBanking
inicio
    const string CLAVEVALIDA = "Clave1234"

    var string clave
    mostrar: "Ingrese su clave"
    ingresar: clave

    si clave = CLAVEVALIDA entonces
        mostrar: "Clave correcta"
    sino
        mostrar: "Clave incorrecta"
    fin si
fin
```

6)

```
programa TipoDeAuto
inicio
    var integer nroDePuertas
    mostrar: "Ingrese cantidad de puertas"
    ingresar: nroDePuertas

    si nroDePuertas = 4 entonces
        mostrar: "Su auto es sedán"
    sino
        mostrar: "Su auto No es sedán"
    fin si
fin
```

7)

```
programa CirculacionPatentePar
inicio
    var integer patente
    mostrar: "Ingrese los tres números de su patente"
    ingresar: patente

    si patente%2 = 0 entonces
        Mostrar: "Esta habilitado para circular"
    sino
        mostrar: "No esta habilitado para circular. Le llegará una multa"
    fin si
fin
```

8)

```
programa MayorDeEdad
inicio
    var integer edad
    mostrar: "Ingrese su edad"
    ingresar: edad
    si edad >= 18 entonces
        mostrar: "Usted es mayor de edad"
    sino
        mostrar: "Usted es menor de edad"
    fin si
fin
```

//ESTRUCTURAS CONDICIONALES ANIDADAS

9)

```
programa DescripcionDeUnaPersonaSegunSuAltura
inicio
    var float altura
    mostrar: "Ingrese Altura"
    ingresar: altura

    si altura <= 1,50 entonces
        mostrar: "Persona de altura baja"
    sino
        si altura < = 1,70 entonces
            mostrar: "Persona de altura media"
        sino
            mostrar "Persona alta"
        fin si
    fin si
fin
```

10)

```
Programa DiezAnios
inicio
    const integer ANIODESDE = 2000
    const integer ANIOHASTA = 2010
    var integer anio

    mostrar: "Ingrese Año:"
    ingresar: anio

    si anio < ANIODESDE entonces
        mostrar: "Año ingresado fuera de rango"
    sino
        si anio <= ANIOHASTA entonces
            mostrar: "Ingreso Permitido"
        sino
            mostrar: "Año ingresado fuera de rango"
        fin si
    fin si
fin
```

11)

```
programa VotaONo
inicio
    var integer edad = 0
    mostrar: "Ingrese su edad"
    ingresar: edad

    si edad >= 18 entonces
        si edad >= 70 entonces
            mostrar: "Usted no tiene la obligación de votar"
        sino
            mostrar: "Usted tiene la obligación votar"
        fin si
    sino
        mostrar: "Usted no puede votar"
    fin si
fin
```

12)

```
programa Calificacion
inicio
    var integer nro
    mostrar: "ingresar un numero de orden"
    ingresar: nro

    si nro <= 2 entonces
        mostrar: "está en la primera fila"
        si nro = 1 entonces
            mostrar: "además, está en la pole position"
        fin si
    sino
        si nro <= 4 entonces
            mostrar: "en segunda fila"
        sino si nro <= 8 entonces
            mostrar: "peloton del medio"
        sino
            mostrar: "saliendo de atrás"
        fin si
    fin si
fin
```

//ESTRUCTURAS CONDICIONALES COMPUESTAS

13)

```
programa CostoDelFlete
inicio
    var float precioCompra
    mostrar: "Ingrese el precio final de la compra"
    ingresar: precioCompra

    si precioCompra >= 2000 entonces
        mostrar: "El flete es gratis"
    sino si precioCompra < 2000 Y precioCompra >= 1000 entonces
        mostrar: "El costo del flete es $150"
    sino
        mostrar: "El costo del flete es $250"
    fin si
fin
```

14)

```
programa SoloEfectivo
inicio
    var string formaDePago
    mostrar: "Como va a pagar?"
    ingresar: formaDePago

    si formaDePago = "tarjeta" entonces
        mostrar: "Lo sentimos, esta semana sólo aceptamos efectivo"
    sino si formaDePago = "efectivo" entonces
        mostrar: "Pase por la caja y le cobran, Gracias"
    sino
        mostrar: "Forma de pago no aceptada"
    fin si
fin
```

15)

```
programa RadioFavorita
inicio
    const string RADIOFAVORITA = "Horizonte 94.3"

    var string radio
    mostrar: "Radio favorita:"
    ingresar: radio

    si radio <> RADIOFAVORITA entonces
        mostrar: "Su respuesta es incorrecta"
    sino si radio = RADIOFAVORITA entonces
        mostrar: "Su repuesta en correcta"
    fin si
fin
```

16)

```
programa DesempenioDelAlumno
inicio
    var integer nota
    mostrar: "Ingresar nota del examen"
    ingresa: nota

    si nota > 0 Y nota < 6 entonces
        mostrar: "Desaprobado"
    sino si nota >= 6 Y nota <= 8 entonces
        mostrar: "Sobresaliente"
    sino si nota > 8 Y nota <= 10 entonces
        mostrar: "Muy sobresaliente"
    sino
        mostrar: "No es una nota valida"
    fin si
fin
```

//ESTRUCTURAS CONDICIONALES MÚLTIPLES

17)

```
programa CodigoDeArea
inicio
    var string codigo
    mostrar: "Ingrese código de área:"
    ingresar: codigo

    en caso de codigo hacer
        caso = "0341"
            mostrar: "Santa Fé"
        fin caso
        caso = "0343"
            mostrar: "Entre Ríos"
        fin caso
        caso = "011"
            mostrar: "Buenos Aires"
        fin caso
        sino
            mostrar: "Código de Área Desconocido"
    fin en caso de
fin
```

18)

```
programa CalificacionMultipleChoice
inicio
    var integer nroDeAciertos
    mostrar: "Ingresar número de respuestas correctas"
    ingresar: nroDeAciertos

    en caso de nroDeAciertos hacer
        caso <= 100 Y >= 90
            mostrar: "Sobresaliente"
        fin caso
        caso < 90 Y >= 70
            mostrar: "Notable"
        fin caso
        caso < 70 Y >= 50
            mostrar: "Bien"
        fin caso
        caso < 50 Y >= 40
            mostrar: "Aprobado"
        fin caso
        caso < 40
            mostrar: "Suspenso"
        fin caso
    fin en caso de
fin
```

19)

```
programa CuantoVoyAGanar
inicio
    var string puesto
    var integer sueldo
    var float sueldoAnual
    var float medioAguinaldo

    const integer CANTMESES = 12

    mostrar: "Por favor ingrese el puesto al que se postula"
    ingresar: puesto

    en caso de puesto hacer
        caso = "Gerente"
            sueldo = 100000
        fin caso
        caso = "Supervisor"
            sueldo = 55000
        fin caso
        caso = "Asistente"
            sueldo = 23000
        fin caso
    fin en caso de

    medioAguinaldo = sueldo * 6 / CANTMESES
    sueldoAnual = CANTMESES * sueldo + 2 * medioAguinaldo

    mostrar: "Su remuneración anual será de $ " + sueldoAnual
    mostrar: "Más un bono por presentismo de $ " + (sueldo / 4)
fin
```

20)

```
programa LasCuatroEstaciones
inicio
    var string letra
    mostrar: "Ingresa una letra"
    ingresar: letra

    en caso de letra hacer
        caso = "P"
            mostrar: "Es primavera"
        fin caso
        caso = "I"
            mostrar: "Es invierno"
        fin caso
        caso = "O"
            mostrar: "Es otoño"
        fin caso
        caso = "V"
            mostrar: "Es verano"
        fin caso
        sino
            mostrar: "No corresponde a ninguna estación"
    fin en caso de
fin
```

ANEXO 2 - Respuesta Actividad de Análisis - Integración

¿Trataron de resolver los requerimientos?

En caso afirmativo, avancen 3 páginas.

En caso negativo, ¡no avancen y traten de resolverlos antes de ver las respuestas!

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

1) PUNTO 1:

A) Forma ideal de resolverlo:

Una variable que vale 100, un contador que resta de a 1, un "mostrar" con el valor de la resta, ambas instrucciones dentro de un ciclo que llega hasta 0.

B) Ejemplos de resolución:

B.1) Se muestra el 100 y luego de inicia el ciclo mostrando desde el 99 hasta el 0, cuando se cumple con la condición de salida del ciclo (0 > 0 es falso y no vuelve a entrar)

```
programa CuentaRegresiva
inicio
    var integer i = 100

    mostrar: i
    mientras i > 0 hacer
        i = i - 1
        mostrar:  i
    fin mientras
fin
```

B.2) Usando 2 variables, se ingresa al ciclo, se muestran el 100 hasta el 0, cuando "maximo = -1" no se cumple con la condición de ciclo: "-1 >=0" es falso

```
programa NumerosDescendentes
inicio
    var integer minimo = 0
    var integer maximo = 100

    mientras maximo >= minimo hacer
        mostrar: maximo
        maximo = maximo - 1
    fin mientras
fin
```

B.3) Se ingresa al ciclo porque 100 no es 0, se resta y muestran los números, que i=1, se vuelve a ingresar al ciclo, "i" se resta y vale 0, se muestra "i" y a la próxima iteración del ciclo no se ingresa porque 0 no es distinto de 0 (0 != 0 es falso)

```
programa ContadorDel100Al0
inicio
    var integer i = 100

    mostrar: i
    mientras i != 0 hacer
        i = i - 1
        mostrar: i
    fin mientras
fin
```

B.4) Similar al segundo caso, misma lógica, pero con una variable menos y un mensaje más.

```
programa ContadorDecreciente
inicio
    var integer i
    i = 100

    mientras i >= 0 hacer
        mostrar : "El valor de i es: "
        mostrar: i
        i = i - 1
    fin mientras
fin
```

B.5) Similar al caso anterior pero sin el mensaje adicional.

```
programa Secuencia100A0
inicio
    var integer numero
    numero = 100

    mientras numero >= 0 hacer
        mostrar: numero
        numero = numero - 1
    fin mientras
fin
```

B.6) Similar al segundo caso, pero usando una constante

```
programa NumerosDecrecientes
inicio
    var integer numInicial = 100
    const integer NUMOBJETIVO = 0

    mientras numInicial >= NUMOBJETIVO hacer
        mostrar: numInicial
        numInicial = numInicial - 1
    fin mientras
fin
```

B.7) Similar al tercer caso, aunque empezando desde el 101, restando primero y luego mostrando todos los números

```
programa CuentaRegresiva
inicio
    var integer nro
    nro = 101

    mientras nro != 0 hacer
        nro = nro - 1
        mostrar: nro
    fin mientras
fin
```

C) Comprobación genérica para todas las alternativas:

C.1) Si o si vamos a necesitar un ciclo "mientras".

C.1.1) Si lo tenemos, seguimos.

C.1.2) Si no lo tenemos, estará mal

C.2) Se está mostrando el 100? Verificar que primero se muestre el 100 y luego se reste y no al revés. Si iniciamos el proceso en 101, verificar que primero se reste y luego de muestre.

C.2.1) Si es verdad, seguimos.

C.2.2) Si no, hay un error

C.3) Se está mostrando el 99? Si es verdad, seguimos.

C.3.1) Si es verdad, seguimos.

C.3.2) Si no, hay un error

C.4) De ahí hasta el 1 seguramente no hay problemas, seguimos

C.4.1) Si es verdad, seguimos.

C.4.2) Si no, hay un error

C.5) Se está mostrando el 1? Si es verdad, seguimos.

C.5.1) Si es verdad, seguimos.

C.5.2) Si no, hay un error

C.6) Si el valor de la variable es 0, lo estamos mostrando? Si es verdad, seguimos.

C.6.1) Si es verdad, seguimos.

C.6.2) Si no, hay un error

C.7) Verificar que no está mostrando el -1. Si es verdad, seguimos.

C.7.1) Si es verdad, seguimos.

C.7.2) Si no, hay un error

C.8) Verificar que no está mostrando más de una vez el 0. Si es verdad, el programa cumple con el requerimiento.

C.8.1) Si es verdad, seguimos.

C.8.2) Si no, hay un error

D) Errores comunes:

- No es necesario ingresar ningún valor. Mucho menos ingresarlo y luego pisarlo con "100" o ingresarlo y no usarlo. Mucho peor empezar a mostrar los números desde un valor ingresado: si ingreso el -422 el programa no va a ser lo que se le pidió que haga

- No se cumple con el alcance si no se están mostrando los valores solicitados dentro del ciclo: se olvidaron de colocar el "mostrar:"

- No se cumple con el alcance si se está mostrando el número 0 dos veces. Muy común, lo muestran dentro del ciclo y luego, fuera, lo vuelven a mostrar. Cualquier agregado o faltante de números enteros en el 100 y el 0 (incluidos, ninguno duplicado) estará mal.

- No se cumple con el alcance si no se está mostrando el 100. Muchos hacen la asignación inicial (i=100) y luego, en el ciclo, primero hacen "i=i-1" y luego el "mostrar:", por lo tanto se muestran los números desde el 99.

- No se cumple con el alcance si se está mostrando los números desde el 100 al -1. Cuando está mal la condición de salida del ciclo, se hace "una vuelta más" generando que se agregue el -1. Como no es uno de los números solicitados, es un error.

- No declarar correctamente el contador: "Total - 1" en lugar de "Total = Total - 1", es un error.

- Desde ya: no pueden hacer 101 "mostrar" (no usar un ciclo repetitivo).

- Errores en la condición de salida del ciclo: casos de loops infinitos (por hacer mal el contador, no hacer el contador, que la variable de la condición de salida del ciclo no sea la variable decrementada, etc...)

2) PUNTO 2:

A) Forma ideal de resolverlo:

Se ingresa un valor a una variable ("ingresar: letra"), se hace una condición múltiple evaluando en cada caso si el valor es "a", "e", "i", "o" o "u". En los 5 "caso", va el mensaje correspondiente. En el "sino" el otro mensaje pedido.

B) Ejemplos de resolución:

B.1) Solución más simple y conservadora: estructura condicional múltiple y mensajes

```
programa ValidarVocal
inicio
    var string letra

    mostrar: "Ingrese una letra."
    ingresar: letra

    en caso de letra hacer
        caso = "A"
            mostrar: "Es una vocal."
        fin caso
        caso = "E"
            mostrar: "Es una vocal."
        fin caso
        caso = "I"
            mostrar: "Es una vocal."
        fin caso
        caso = "O"
            mostrar: "Es una vocal."
        fin caso
        caso = "U"
            mostrar: "Es una vocal."
        fin caso
        sino
            mostrar: "No es una vocal."
    fin en caso de
fin
```

B.2) Variante de solución anterior, verificando si la letra ingresada es vocal minúscula o mayúscula

```
programa BuscarVocales
inicio
    var string letra
    mostrar: "Ingrese una letra"
    ingresar: letra

    en caso de letra hacer
        caso ="a" o ="A"
            mostrar: "Es una vocal"
        fin caso
        caso ="e" o ="E"
            mostrar: "Es una vocal"
        fin caso
        caso ="i" o ="I"
            mostrar: "Es una vocal"
        fin caso
        caso ="o" o ="O"
            mostrar: "Es una vocal"
        fin caso
        caso ="u" o ="U"
            mostrar: "Es una vocal"
        fin caso
        sino
            mostrar: "No es una vocal"
    fin en caso de
fin
```

B.3) Alternativa usando constantes y estructura condicional compuesta

```
programa EsVocal
inicio
    var string valorIngresado
    const string VOCALA = "a"
    const string VOCALE = "e"
    const string VOCALI = "i"
    const string VOCALO = "o"
    const string VOCALU = "u"

    mostrar:"Ingrese una letra en minuscula"
    ingresar: valorIngresado

    si valorIngresado=VOCALA entonces
        mostrar: "Vocal"
    sino si valorIngresado=VOCALE entonces
        mostrar: "Vocal"
    sino si valorIngresado=VOCALI entonces
        mostrar: "Vocal"
    sino si valorIngresado=VOCALO entonces
        mostrar: "Vocal"
    sino si valorIngresado=VOCALU entonces
        mostrar: "Vocal"
    sino
        mostrar: "No vocal"
    fin si
fin
```

B.4) Variante del primer caso, con un estructura condicional doble adicional

```
programa EsVocalONoLoEs
inicio
    var string letra = ""
    var boolean esVocal = falso
    const string mensajeEsVocal = "La letra ingresada es una vocal"
    const string mensajeNoEsVocal = "La letra ingresada es una vocal"

    mostrar: "Ingrese una letra"
    ingresar: letra

    en caso de letra hacer
        caso = "a"
            esVocal = verdadero
        fin caso
        caso = "e"
            esVocal = verdadero
        fin caso
        caso = "i"
            esVocal = verdadero
        fin caso
        caso = "o"
            esVocal = verdadero
        fin caso
        caso = "u"
            esVocal = verdadero
        fin caso
    fin en caso de

    si esVocal = verdadero entonces
        mostrar: mensajeEsVocal
    sino
        mostrar: mensajeNoEsVocal
    fin si
fin
```

B.5) Variante de la primera solución, utilizando intensivamente constantes

```
programa EsVocalONoEsVocal
inicio
    var string letra
    const string VOCAL1 = "a"
    const string VOCAL2 = "e"
    const string VOCAL3 = "i"
    const string VOCAL4 = "o"
    const string VOCAL5 = "u"
    const string VALIDO = "Esta es una vocal"
    const string ERROR = "Esta no es una vocal"

    mostrar: "Ingrese una letra"
    ingresar: letra

    en caso de letra hacer
        caso = VOCAL1
            mostrar: VALIDO
        fin caso
        caso = VOCAL2
            mostrar: VALIDO
        fin caso
        caso = VOCAL3
            mostrar: VALIDO
        fin caso
        caso = VOCAL4
            mostrar: VALIDO
        fin caso
        caso = VOCAL5
            mostrar: VALIDO
        fin caso
        sino
            mostrar: ERROR
    fin en caso de
fin
```

C) Comprobación genérica para todas las alternativas:

Si no evaluaron las 5 opciones ("a", "e", "i", "o", "u"), si hay demasiados "si/fin si", si no hay al menos una estructura condicional y si no hay un solo "ingresar:", el programa va a estar mal.

D) Errores comunes:

- "Vocal" no es una instrucción válida. Una computadora no puede saber lo que para Ustedes es una vocal o una consonante si no se lo definen antes. Por ejemplo:

```
var string letra

ingresar: letra

si letra = vocal entonces       //qué es "vocal"??
```

- "letras = aeiou" no es una instrucción válida. No es posible asignar más de un valor a una variable. Por ejemplo:

```
var string miLetra

var string letras = aeiou       //operación inválida

ingresar: miLetra

si miLetra = letras entonces
```

- "si letra = aeiou entonces" o "si letra = a,e,i,o,u entonces" y variantes, no es una instrucción válida, dado que se tiene que aclarar qué variable se compara contra qué valor. Por ejemplo, forma correcta:

```
si letra = "a" O letra = "e" O letra = "i" O letra = "o" O letra = "u" entonces
```

- La opción sugerida sería usar una estructura "en caso de", dado que están evaluando una variable contra más de 3 o 4 valores posibles. Condiciona compuesta también es alternativa. Se debe evitar la anidadas o dobles, en este caso.

- Mal uso de la estructura condicional múltiple: la variable va únicamente en el "en caso de .... hacer". NO va en el "caso", donde sólo va al condición. Por ejemplo, forma incorrecta:

```
en caso de letra hacer
        caso letra = "a"
                //...
        fin caso
        caso letra = "e"
                //...
        fin caso
//...
```

Ejemplo, forma correcta:

```
en caso de letra hacer
        caso = "a"
                //...
        fin caso
        caso = "e"
                //...
        fin caso
//...
```

- Mezclar las estructuras condicionales. Nunca puede haber un "caso" dentro de un "si". Sí una estructura "en caso de" completa, pero no "caso" sueltos. El "caso" va ÚNICAMENTE en la estructura "en caso de".

- Intentos de usar vectores como "vocal[letra] {letra = (A,E,I,O,U)}" o similares, no son necesarios ni han sido hasta esa unidad.

3) Errores comunes en ambos puntos

- No colocar el bloque "programa" antes del "inicio".

- Las asignaciones deben hacerse con el "=", NUNCA con ":"

- No declarar variantes o constantes de usarlas (palabras reservadas "var" y "const") o no indicar el tipo de datos (string, integer, boolean, float, double, date).

- Usar variables (en comparaciones) sin previamente asignarle o ingresarle un valor (nunca se puede comparar algo que tiene valor contra algo que no tiene valor, es una igualdad que está destinada a ser falsa siempre)

ANEXO 3 - Checklist de código

Algunas consideraciones a la hora de revisar su propio código que se suman al checklist de la unidad anterior:

	Cumple	
Tema	**Si**	**No**
Todas las variables están declaradas antes de ser usadas		
Todas las constantes están declaradas antes de ser usadas		
Todas las constantes tienen un valor asignado antes de ser usadas		
Las contantes mantienen su valor inicial en todo el flujo del programa		
Las variables o constantes que van a ser usadas en una comparación, tiene valor asignado o ingresado?		
Las variables o constantes que van a ser usadas en un ciclo repetitivo, tiene valor asignado o ingresado?		
Las variables o constantes de las cuales se va a mostrar su valor, tiene valor asignado o ingresado?		
Todas las constantes y variables, en su declaración, tienen asignado un solo tipo de dato (string, integer, boolean, float, double, date)		
Todos los "si" tienen su "fin si"		
Todos los "caso" tiene su "fin caso"		
Todos los "en caso de" tienen su "fin en caso de"		
Todos los "mientras" tienen su "fin mientras"		
Todas los "mientras" tienen alguna forma de terminar (evitando el loop infinito) modificando DENTRO del ciclo al menos una de las variables que conforman su condición de salida		
Todos los "si" tienen su "entonces"		
Se evitó el uso de palabras reservadas inexistentes como "fin sino"		
Todos los contadores y acumuladores están guardando su valor nuevo es una de las variables utilizadas en la operación		
Todas las variables usan el estándar de nombrado lowerCamelCase		
Todas las constantes usan el estándar de nombrado UPPER CASE		
El código está compuesta ÚNICAMENTE por las palabras reservadas (programa, inicio, fin, ingresar, mostrar, var, const, string, integer, boolean, float, double, date), variables, valores y operaciones válidas		
Todas las variables de tipo string a las que se le asigna un valor, tienen ese valor entre comillas dobles ("")		
Todas las variables de tipo boolean tiene asignado únicamente el valor "verdadero" o "falso"		
Todas las variables de tipos numéricos (integer, float, double) a las cuales se les hace una asignación no tienen comillas en su valor asignado		

Todas las variables a las que se les asigna un valor solo recibien UN valor		
Las comparaciones de valores entre variables se hacen siempre con variables del mismo tipo de dato		
Las comparaciones de valores entre constantes se hacen siempre con constantes del mismo tipo de dato		
Las comparaciones entre variables y constantes se hacen siempre con las variables y constantes del mismo tipo de dato		
Las asignaciones de valores entre variables, constantes y combinación de ambos, siempre se hace contemplando que deben tener el mismo tipo de dato		

Si todas las respuestas son afirmativas, hay muy altas chances que el programa esté libre de errores de código.

ANEXO 4 - Preguntas frecuentes

1) Pregunta: En las condiciones múltiples, en algunas ocasiones se usa la expresión "caso es..." y en otras no.

caso es < 0

caso es > 0

Y

caso = 0

¿Cuál es la forma correcta?

Respuesta: "caso es..." o "caso..." es indistinto en pseudocódigo, se puede usar cualquiera de las dos variantes. La premisa, usando pseudocódigo, siempre será seguir los lineamientos generales y que se sea claro. Al no compilarse ni ejecutarse, podemos obviar la parte del cumplimiento estricto de sintaxis de los lenguajes de programación.

2) P: ¿Por qué en uno de los ejemplos aparece una coma (,) entre comillas?

mostrar: "el nombre y clave ingresado son" + nombre + "," + clave

R: La coma forma parte de la cadena de caracteres de la instrucción "mostrar:", por eso va entre comillas y adyacente se coloca el operador "+" para poder concatenar las cadenas de caracteres. En otras palabras, "," es un valor fijo.

3) P: Si cuento con un programa que solicita a un usuario el ingreso de tres valores que representan las calificaciones de una materia, pero no sabemos si dicha calificaciones son un número entero o decimal, convendría usar un tipo de dato "float" directamente?

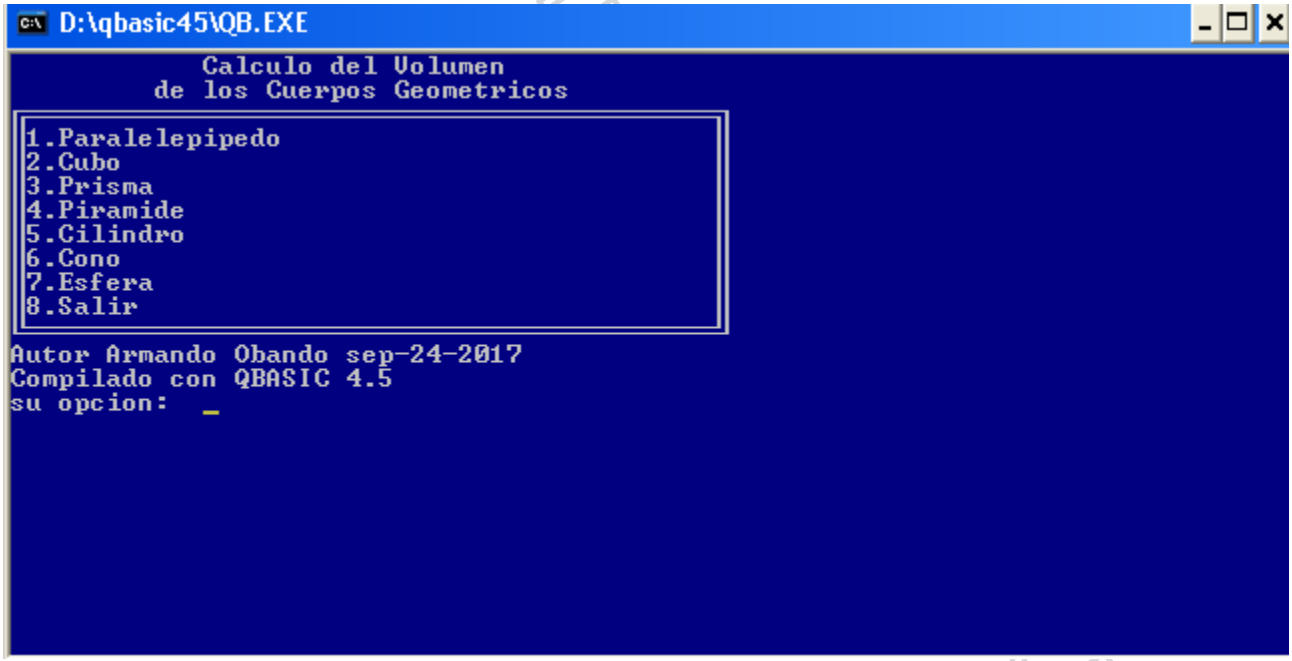
R: Es muy probable que ese dato necesite un tipo numérico decimal. Como los números decimales también permiten números enteros (,00), ante la duda, es conveniente usar tipos de datos decimales.

4) P: Un programa puede tener consultas a bases de datos, diseño de botones, tablas, cajas de textos, etc. ¿Todas esas declaración y procesos son representables con pseudocódigo?

R: Ni la persistencia de datos ni el diseño de interfaz de usuario hacen a los principios de la lógica computacional, por lo tanto no es posible ni tiene sentido representarlos con pseudocódigo ni están incluidos en el temario del curso.

5) P: En un salida de datos ("mostrar:"), los valores deben ser mostrados de a uno o todos juntos?

R: No importa cómo se muestren, pero se tienen que mostrar todos. Si quisieran visualizar mentalmente la "ejecución" de un problema en pseudocódigo, que no cuenta con interfaz de usuario ni elementos gráficos, se podría asimilar, por ejemplo, a la ejecución por línea de comandos de un programa en C, Clipper o, incluso, QBasic, como se ve en la imagen.



(Tomado de lawebdelprogramador.com - https://www.lawebdelprogramador.com/codigo/Basic/index2.html)

6) P: ¿Se puede hacer una operación de este tipo en una declaración?

var float valorEslora

var float valorManga

var float valorPuntal

*var float valorTat = valorEslora*valorManga*valorPuntal/5 // (*)*

R: Lo que está marcado con (*) no se puede hacer, por una simple razón: ni "valorEslora", ni "valorManga", ni "valorPuntal" tienen valor asignado o ingresado al momento de hacer la operación, por lo que esa multiplicación en algunos lenguajes será cero y en otros será un error en tiempo de ejecución por intentar operar con valores nulos.

7) P: ¿En el caso de usar constantes en una condición, digamos, la constante "EJEMPLO" y tener que comparar su valor contra variables o valores, al momento de armar el condicional, es posible hacer este tipo de condiciones?

si EJEMPLO != dato1, dato2 entonces...

O

si EJEMPLO != dato1 o dato2 entonces...

R: No es correcto hacer "EJEMPLO != dato1, dato2, dato3"

Siempre se debe contar con pares ordenados, del tipo:

"VALOR1 + OPERACION_LOGICA VALOR2"

Esto, que podemos definir como un "término" de la condición se puede unir con otros términos a través de los operadores "Y" u "O". En base a estos operadores se va a saber si se tienen que cumplir todos o algunos de los términos para ingresar dentro de la condición verdadera del condicional.

Entonces, la forma correcta sería:

si EJEMPLO != valor1 o EJEMPLO != valor2 entonces...

Esto aplica tanto para constantes como para variables.

8) P: En alguno de los ejemplos de condicionales la condición dice:

si nombre = nombreGuardado entonces

¿Ese igual ("=") cambia su función en el condicional? ¿Por qué se usa para comparar y no para asignar?

R: El símbolo "=" en una estructura condicional es un operador lógico, no de asignación. Los operadores tienen un funcionamiento contextual, en todos los lenguajes, por lo tanto puede cambiar su función dependiendo del uso que se le esté dando. Lo mismo ocurre con el operador "+" que sirve tanto para sumar (si está entre dos números) como para concatenar (cuando no está entre dos números).

9) P: En lo que hace a las estructuras repetitivas, la inicializacion de acumuladores y contadores ¿en todo los casos se tienen que hacer afuera del ciclo?

R: Si: siempre afuera del ciclo.

Si la idea es contar o acumular (cualquier cosa, en la vida real) se tiene que empezar por algún lado, un punto de partida, normalmente el cero.

Cuando se cuentan los goles de un partido de fútbol que está iniciando, por ejemplo, el marcador se inicia en cero o en 15.632? En cero. ¿En qué minuto comienza el partido? En el minuto cero.

Por lo tanto, las variables usadas en contadores o acumuladores se inicializan siempre afuera de la estructura repetitiva.

10) P: ¿Es posible declarar un variable así?

const string vocal = "a" o "e" o "i" o "o" o "u"

R: No, una variable solo puede guardar un valor. Ni operaciones, ni varias valores, ni tipos de datos distintos.

11) P: Cuando se menciona en las estructuras condicionales múltiples que "No debería evaluarse más de un valor en cada "caso", ya que se pierde el sentido de prolijidad que aporta la estructura condicional múltiple "en caso de"... ¿Quiere decir que no es posible hacer algo así?

caso =2 O caso =5

caso >5 Y <10

R: No se "debería", pero como poder, se puede. Pero es verdad que se pierde parte de la claridad que aporta la estructura múltiple.

12) P: ¿Qué es un módulo? ¿Por qué las variables se declarar al principio de cada módulo?

R: La implementación concreta de la modularización está dada por el concepto de "funciones" que se presentará próximamente.

De momento, el único bloque de código que tenemos definido hasta ahora es el "inicio"/"fin": una buena práctica sería declarar las variables inmediatamente luego del "inicio".

13) P: Se podría asignar a una variable o constante un valor que incluya ><?

Por ejemplo:

const integer litros = <10

const integer litros = >0 <10

R: No, no es posible. Las variables y constantes almacenan un único valor, no conjuntos de valores ni operaciones lógicas.

14) P: Los tipos de datos booleanos: ¿son siempre verdaderos o falsos, o puede responder a otros valores, como: si/ no, o encendido/ apagado?

R: Tomamos como valores posibles de tipos de datos booleanos únicamente "verdadero" y "falso".

15) P: Con las palabras reservadas que hemos definido hasta el momento ¿existe alguna manera de controlar si el valor ingresado es una letra o un número?

R: No, el pseudocódigo no ofrece ningún mecanismo de validación de tipos de datos.

16) P: ¿En qué momento se debe/puede/conviene inicializar una variable del tipo boolean? ¿Al inicio del programa o justo antes de la estructura condicional dónde se utiliza?

R: Siempre debe declararse antes de ser usada, preferentemente al inicio del programa (con esto nos aseguramos lo anterior). Y no importa el tipo de dato.

Con este criterio, si la variable necesita recibir un valor fijo por parte del programador para que el programa funcione bien, se puede hacer durante o luego de la declaración.

Ejemplo 1:

```
var string apellido

apellido = "lopez"

si apellido = "gomez" entonces...
```

Ejemplo 2:

```
var string apellido = "lopez"

si apellido = "gomez" entonces...
```

Si un usuario va a ingresar un valor en esa variable, no es necesario inicializarla.

Ejemplo:

```
var string apellido

ingresar: apellido
```

17) P: Si, por ejemplo, una variable o constante fuera inicializada al principio:

var string apellido = "lopez"

¿El usuario igualmente debe o puede ingresar un nuevo valor en esa variable?

R: Ante todo, hay que tener en claro qué se está haciendo:

Sabiendo que ingresar o asignar implica darle un valor a una variable. ¿para qué se está haciendo esa acción? ¿Dónde vas a usar el valor? ¿Por qué? ¿El valor que va a tener la variable, lo conoce el programador o es un valor desconocido que debe ser ingresado por el usuario?

No es una cosa o la otra y la misma regla siempre. Las instrucciones hay que darlas según lo que se NECESITE hacer, a conciencia, haciendo lo justo y necesario.

Dicho esto, no tendría sentido hacer algo como:

var string apellido = "lopez"

ingresar: apellido

La forma correcta sería:

var string apellido = ""

ingresar: apellido

Aunque la inicialización (= "") no es obligatoria en este caso, porque no se usa la variable ni se consulta su valor antes de que reciba un valor.

Lo que vimos

- Principales aspectos del paradigma estructurado de programación
- Uso y funcionalidad de las variables
- Uso y funcionalidad de las constantes
- Uso de tipos de datos para variables y constantes
- Estructuras de control y sus principales variantes
- Otras estructuras comunes

Lo que viene:

- Conceptos y uso de los vectores y matrices
- Uso de funciones y conocer los tipos