Curso de Algoritmos y Estructura de Datos

PRIMERA PARTE

UNI FIIS

THE OLIZTIK

Este pequeño compendio va dirigido a los estudiantes de la Facultad de Ingeniería Industrial y de Sistemas de la Universidad Nacional de Ingeniería que se encuentren llevando el curso de Algoritmos y Estructura de Datos.

En el tiempo que llevo en la universidad he podido notar que "algoritmos y estructura de datos" es uno de los cursos con el mayor número de desaprobados y creo que esta situación debería mejorar. Es por esto que he elaborado este compendio de la primera parte del curso que contiene los temas hasta la primera práctica calificada (conceptos básicos, los datos simples, y las estructuras de control). Los ejercicios presentados aquí fueron extraídos de distintas fuentes: Algunos corresponden a los resueltos en clase, prácticas calificadas, libros, como también de un texto denominado "cuaderno de algoritmo" que encontré en el siguiente enlace **www.ingfiis.tk** en la sección "**2do ciclo**" y de apuntes que he realizado mientras llevé el curso. La teoría presentada en este compendio es resumida y se enfoca en la parte práctica, así que si quisieran profundizar en el tema les recomendaría el libro "Fundamentos de Programación, algoritmos y estructura de datos y objetos" de Luis Joyanes Aguilar.

Todos los ejercicios de este texto deben de ser comprendidos y entendidos perfectamente. Al principio tal vez les parezcan complicados o confusos (eso nos ha sucedido a todos en algún momento) pero mientras más practiquen irán dominándolos. Como ya les había mencionado, por ahora solo les puedo presentar la primera parte del curso. Luego de esta primera parte estudiarán temas como "arreglos", "cadenas", "registros", etc... en los que de todas maneras seguirán haciendo uso de las técnicas ya aprendidas en este texto. Es por esto que creo que el dominio de este compendio es importante.

Algunas sugerencias:
Hasta la fecha, los profesores que regularmente enseñan el curso son: Inga, Acosta, Grimanesa y Córdova.
La mayoría de estudiantes que llevaron el curso con la profesora Inga comentan que es ordenada y explica muy bien, aunque últimamente me han comentado que ha comenzado a explicar mediante diapositivas. El profesor Acosta es ordenado, es puntual en la revisión de los exámenes, es muy estricto en su calificación, avanza el curso conforme el cronograma pero no explica muy bien y tiene la costumbre de realizar problemas difíciles cuando debería comenzar resolviendo ejercicios sencillos para poder entender lo esencial del asunto.
La profesora Grimanesa es bien organizada en su clase pero eso ocasiona que muchas veces se retrase, además es flexible en su calificación, generalmente, los alumnos que se matriculan en su sección son repitentes (no se ofendan) que ya tienen la base del curso y que desean aprobar.
El profesor Córdova es un ingeniero con mucha experiencia y domina la materia, pero sus clases en la pizarra son poco ordenadas.

Por otro lado deben de saber que el formato que utilizarán para la resolución de los problemas variará según la sección en donde se encuentren matriculados. ¿Por qué? Porque para la resolución de los problemas

no existe un formato estándar, sino que cada persona tiene un estilo propio, así que deberán aprender el estilo usado por sus profesores.

Para la resolución de los problemas de este texto usaré el formato del profesor Acosta porque es un estilo muy completo y fácil de entender. Muy aparte del formato también deben tener presente que un mismo problema puede resolverse de diferentes maneras, en la mayoría de los ejemplos de este texto solo se presenta una forma de resolución, pero eso no quiere decir que sea la única.

Si quisieran colaborar con ejercicios interesantes, resoluciones de prácticas y exámenes pueden escribirme al correo electrónico theoliztik@gmail.com. Espero recibir toda clase de aportes así como comentarios, correcciones y críticas para que entre todos podamos ir mejorando este texto que espero sea de mucha ayuda.

The Oliztik

25 de Agosto 2011

LISTADO DE TEMAS

UNIDAD I
INTRODUCCIÓN

1. ALGORITMOS

- **1.1. ALGORITMOS EN LA COMPUTACIÓN**
- **1.2. FASES EN LA RESOLUCIÓN DE PROBLEMAS**
  - **1.2.1 ANÁLISIS DEL PROBLEMA**
  - **1.2.2 DISEÑO DE LA SOLUCIÓN**
- **1.3 CARACTERÍSTICAS DE UN ALGORITMO**
  - **1.3.1 CARACTERÍSTICAS NECESARIAS DE UN ALGORITMO**
  - **1.3.2 CARACTERÍSTICAS RECOMENDABLES DE UN ALGORITMO**
- **1.4. TÉCNICAS DE REPRESENTACIÓN DE UN ALGORITMO**
  - **1.4.1 DIAGRAMA DE FLUJO**
  - **1.4.2 PSEUDOCÓDIGO**
  - **1.4.2 PSEUDOCÓDIGO**

2. CONCEPTOS BÁSICOS

- **2.1. DATO E INFORMACIÓN**
  - **2.1.1. DATO:**
  - **2.1.2. INFORMACIÓN:**
- **2.2. ATRIBUTOS DE UN DATO**
  - **2.2.1 PRIMER ATRIBUTO: TIPO**
  - **2.2.2. SEGUNDO ATRIBUTO: VALOR**
  - **2.2.3. TERCER ATRIBUTO: IDENTIFICADOR**

3. CONSTANTES Y VARIABLES

- **3.3.1 MEMORIA**
- **3.3.2 DECLARACIÓN DE UNA VARIABLE**
- **3.3.3. ACCESO A LA CELDA DE MEMORIA RESERVADA**
- **3.1 CONSTANTES**
- **3.2 VARIABLES**
- **3.3 LA VARIABLE COMO UNA POSICIÓN DE MEMORIA**

4. OPERADORES Y EXPRESIONES

- **4.1 OPERADORES ARITMÉTICOS Y EXPRESIONES ARITMÉTICAS**
  - **4.1.1 OPERADORES ARITMÉTICOS**
  - **4.1.2 EXPRESIONES ARITMÉTICAS**
- **4.2 OPERADORES RELACIONALES O DE COMPARACIÓN Y EXPRESIONES RELACIONALES**
  - **4.2.1 OPERADORES RELACIONALES**
  - **4.2.2 EXPRESIÓN RELACIONAL**
- **4.3 OPERADORES LÓGICOS Y EXPRESIONES LÓGICAS**
  - **4.3.1 OPERADORES LÓGICOS**
  - **4.3.2 EXPRESIÓN LÓGICA**

5. INSTRUCCIONES

- **5.1 INSTRUCCIÓN DE ASIGNACIÓN**

UNIDAD I

INTRODUCCIÓN

Una de las muchas cosas importantes que van a aprender durante su etapa universitaria es la creación de programas que permitan resolver problemas mediante el uso de una computadora. Un programa de computadora está constituido por un conjunto de instrucciones que están ordenadas y enlazadas de forma coherente tal que al ejecutarse realizan una o varias tareas los cuales permiten la solución de un problema en particular. Para llegar a crear un programa se realizan etapas previas: **Análisis** del problema y el **Diseño** de la solución. Estas dos fases son las que se estudiarán con mucho detalle durante el desarrollo del presente curso. El análisis del problema consiste en comprender el problema y saber **QUÉ** es lo que se espera obtener. Seguidamente el diseño de la solución consiste en construir la forma **CÓMO** se resolverá el problema, es decir, realizar el procedimiento necesario para obtener la solución. Este procedimiento está constituido por un conjunto de pasos lógicos a lo que se le denomina **algoritmo**.

Los algoritmos pueden representarse de varias maneras, ya sea mediante el uso de fórmulas, diagramas de flujo, diagramas Nassi Schneiderman o pseudocódigo. Este último será la representación usada en el curso.

El pseudocódigo como su mismo nombre lo indica es un código falso que luego deberá ser traducido a un lenguaje de programación. El pseudocódigo no puede ejecutarse en una computadora, su escritura se realiza en un papel ya que solo es un boceto del código original.

Una vez finalizado el diseño de la solución (el algoritmo representado mediante pseudocódigo) se procede recién a la fase de codificación en un lenguaje de programación. La fase de codificación se estudia con detalle en los ciclos siguientes.

EN CONCLUSIÓN

Por ahora el trabajo será a base de lápiz y papel, y la preocupación estará principalmente en la ***lógica*** de los procedimientos y en la adopción de algún formato o ***estilo*** (ordenado) para el diseño de los algoritmos. Por lo tanto, por el momento no será necesario aprender la sintaxis de ningún lenguaje de programación en particular, sino lo que se aprenderá será a escribir en pseudocódigo el cual tiene la ventaja de ser independiente pues puede implementarse en cualquier lenguaje de programación. (En algunos con más facilidad y en otros haciendo ligeras modificaciones.)

Tener muy claro lo siguiente: ***¡¡¡ SIN ALGORITMO NO HAY PROGRAMA ¡!!***

Objetivo del Curso: Aprender a pensar como un programador.

¿Por qué? Porque como futuros programadores, nuestra manera de pensar tiene que ser diferente a la de las demás personas al momento de resolver un problema computable.

1. ALGORITMOS

Podemos decir que un algoritmo es un conjunto limitado de **pasos** lógicos que permiten resolver un problema. Las personas continuamente seguimos pasos o desarrollamos una serie de acciones para resolver algún problema o para satisfacer una necesidad. Así que todos consciente e inconscientemente estamos realizando algoritmos.

Ejemplos:

1. *Para llegar a la universidad debemos: levantarnos, asearnos, desayunar, ir al paradero, abordar el bus, etc.*
2. *Al amarrarnos los zapatos realizamos una serie de movimientos ordenados.*
3. *Al vestirnos.*
4. *El conjunto de acciones al momento de realizar una compra en un supermercado.*

El concepto de **algoritmo** es muy general y se encuentra presente en las diferentes ramas del conocimiento. La rama del saber que mayor utilización ha hecho de los algoritmos es la matemática con la que estamos muy relacionados. Ejemplos hay muchos: Sumas de fracciones, al aplicar el **teorema de Pitágoras**, los pasos que realizamos para calcular el máximo común divisor de dos números enteros positivos el cual conocemos como **algoritmo de Euclides**. Además tenemos el **algoritmo de Gauss-Jordan** para resolver sistemas de ecuaciones y que se aprende en el curso de Álgebra Lineal; así como el método general que usamos en la resolución de ecuaciones polinomiales de segundo grado o el algoritmo para encontrar los números primos conocido como **Criba de Eratóstenes,** etc**.**

1.1. ALGORITMOS EN LA COMPUTACIÓN

Los programas de computadora surgen como necesidad de resolver problemas pero son los Algoritmos la base de la programación de computadoras pues detrás de todo programa hay un algoritmo que ha sido previamente codificado en un lenguaje entendible por la computadora. Estos algoritmos (previamente codificados) le ordenan a la computadora las acciones que debe realizar para solucionar el problema en cuestión. Por todo esto, debemos tener **SIEMPRE** presente que para poder realizar cualquier programa es necesario elaborar previamente un algoritmo que contenga los pasos necesarios que resuelvan el problema. Los pasos que forman parte de un algoritmo serán llamados **ACCIONES.**

1.2. FASES EN LA RESOLUCIÓN DE PROBLEMAS

El diseño de algoritmos forma parte de un conjunto de fases que todo programador realiza al momento de solucionar un problema mediante la computadora. Estas fases son básicamente 6:

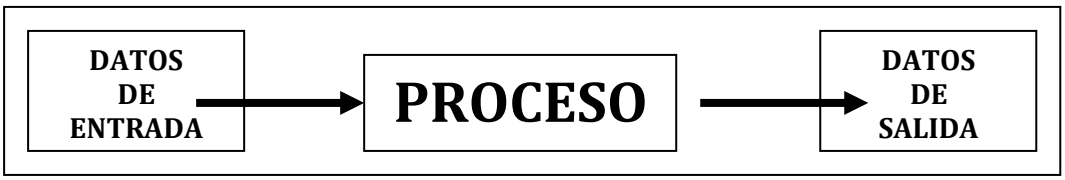
- Análisis del problema.
- Diseño del algoritmo (**Algoritmo**).
- Codificación en un Lenguaje de Programación.
- Compilación y ejecución.
- Verificación y depuración.
- Documentación.

1.2.1 ANÁLISIS DEL PROBLEMA

Lo que hacemos la mayoría cuando estamos ante a una situación problemática es analizarla y tratar de buscar una solución. En eso consiste esta fase, comprender el problema y saber exactamente **qué** es lo que se desea obtener (aunque aún no sepamos cómo hacerlo), luego buscar las alternativas de solución.

Las siguientes interrogantes nos permitirán el correcto análisis de un problema:

1. ¿Con qué información contamos? = ¿Cuáles son los datos de entrada?
2. ¿Qué nos piden como resultado? = ¿Cuáles son los datos de salida?
3. ¿Cuáles son los procesos que transforman los datos de entrada en la información requerida? = ¿De qué forma usamos la datos iniciales para presentar el resultado pedido?



***Ejemplo*: *Resuelve la ecuación polinomial de segundo grado:* $x^2-5x+6=0$**

Solución: Analicemos:

- **DATOS DE ENTRADA**: Tenemos la expresión polinomial.
- **DATOS DE SALIDA**: Debemos presentar la(s) solución(es) de la ecuación.
- **PROCESO**: El análisis podría ser de la siguiente manera:

- Se trata de ecuación polinomial de segundo grado.
- Por el Teorema Fundamental del Álgebra podemos asegurar que tiene dos raíces.
- Si la raíz es de multiplicidad 2, la solución será única.
- Si las raíces son diferentes, habrán dos soluciones.
- Podemos usar la Fórmula general.
- Podemos factorizar mediante aspa simple.
- Podemos formar cuadrados.

En los primeros 4 pasos hemos comprendido el problema. En los últimos pasos estamos determinando **QUÉ** podemos hacer para solucionar el problema. Solo nos quedaría elegir una de estas tres alternativas.

1.2.2 DISEÑO DE LA SOLUCIÓN

Esta es la etapa en la que vamos a solucionar el problema en cuestión mediante la elaboración o diseño de un **algoritmo**, es decir, vamos a especificar paso a paso y detalladamente **CÓMO** obtenemos la solución. Para esto se hace uso de algunas técnicas algorítmicas.

Se puede ordenar el diseño de la siguiente manera:

a) Construir en detalle cada operación.
b) Simplificar antes de calcular.
c) Realizar un dibujo o diagrama. (opcional)
d) Dar una respuesta completa.
e) Hallar el mismo resultado de otra manera. (opcional pero recomendable)

En conclusión: un algoritmo es la solución al problema. Además podemos percibirlo matemáticamente como una función que asocia a cada entrada E una salida S.

Algoritmo: **E → S** / **f (E)=S**

1.3 CARACTERÍSTICAS DE UN ALGORITMO

1.3.1 CARACTERÍSTICAS NECESARIAS DE UN ALGORITMO

1. Debe ser **finito** en tamaño y tiempo por lo que en algún momento debe terminar.
2. Es **preciso** pues los pasos que lo conforman deben estar **ordenados** de manera que no genere ambigüedad. (debe tener una secuencia lógica). **Legibilidad.**
3. Debe de estar bien **definido**, de tal forma que si se sigue dos veces usando los mismos datos, se obtendría el mismo resultado cada vez.
4. Es **Independiente** del lenguaje de programación en el que será implementado. (codificación)

1.3.2 CARACTERÍSTICAS RECOMENDABLES DE UN ALGORITMO

1. Debe ser **Eficiente y Óptimo**, es decir, tiene que solucionar un problema de la mejor forma posible en el menor tiempo.

Ejemplo: Desarrolle un algoritmo para calcular el promedio de tres notas dadas como dato.

Solución:

- DATOS DE ENTRADA: Las tres notas.

- DATOS DE SALIDA: El promedio de las notas.

- PROCESO: Calculamos la media aritmética de las notas.

ALGORITMO:

1. Proporcionar las notas
2. Sumar las tres notas
3. Dividir el resultado entre 3
4. Mostrar el promedio resultante

1.4. TÉCNICAS DE REPRESENTACIÓN DE UN ALGORITMO

Como ya hemos visto, el algoritmo vendría a ser la solución de un problema. Esta solución debe de ser representada de alguna manera y para esto existen diversas técnicas o herramientas. Las técnicas más usadas para la representación de algoritmos son 3:

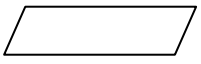
1. **Diagramas de Flujo u Ordinogramas.**
2. **Diagramas Nassi Schneiderman.**
3. **Pseudocódigo**.

Los algoritmos también se pueden representar mediante el **lenguaje natural** pero no resulta eficiente a comparación de las tres que ya se han mencionado además que en muchos casos resulta ambiguo y tiende a interpretarse erróneamente.

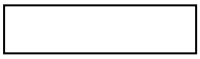
1.4.1 DIAGRAMA DE FLUJO (FLOWCHART)

Es una técnica que permite la representación gráfica de un algoritmo. Esta técnica permite visualizar de forma clara el flujo de los datos y la secuencia de ejecución de las acciones. Utiliza un conjunto de figuras que reciben el nombre de **cajas**. Dentro de las cajas se escriben las acciones que se van a realizar. El conjunto de cajas se conectan mediante líneas llamadas **líneas de flujo**. Estas líneas indican el orden en el que se van a realizar las acciones.

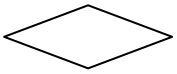
Los símbolos o cajas más utilizadas son:



Entrada / Salida



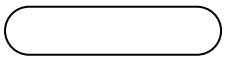
Proceso



Decisión

Línea de Flujo

Conector de diagramas

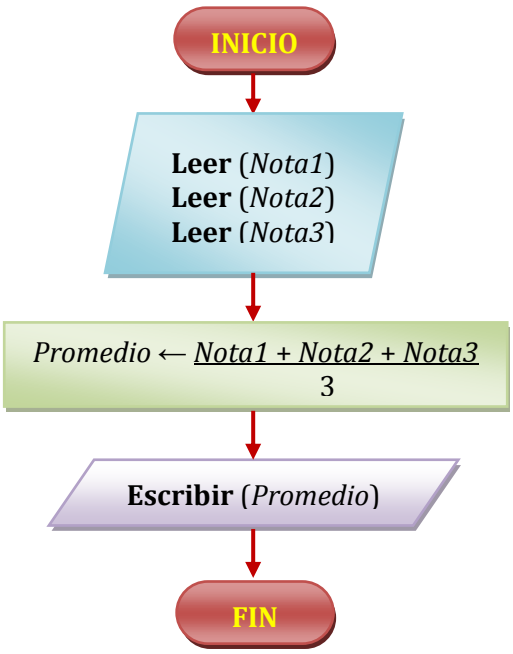


Inicio / Fin

El diagrama de flujo es una de las técnicas más antiguas para representar un algoritmo. Los símbolos usados para la elaboración de diagramas de flujo han sido **normalizados**. La ISO (Organización Internacional para la Estandarización) y el ANSI (Instituto Norteamericano de Estandarización) se encargaron de la estandarización en el año 1985 con la finalidad de evitar que se usen diferentes figuras para representar un mismo proceso lo cual llevaría a confusión.

Ejemplo: *Desarrolle un algoritmo para calcular el promedio de tres notas dadas como dato. Represente el algoritmo mediante un diagrama de flujo*.

Solución:



1.4.2 PSEUDOCÓDIGO

También llamado **Pseudolenguaje.** Es la **representación** escrita de un algoritmo mediante un lenguaje parecido al habla humano. El pseudocódigo es un borrador, boceto, bosquejo, esbozo, etc. Es una simplificación del código original, de allí su nombre: Pseudo-Código = Código Falso.

El pseudocódigo es semejante a un lenguaje de programación, solo que no puede ser ejecutado por una computadora y se rige por normas que no son estándares y que no son tan estrictas como sí ocurre en un lenguaje de programación en particular. Esta similitud con los lenguajes de programación permite que su posterior implementación sea ***relativamente*** fácil.

En términos simples lo podemos concebir como una imitación de algún lenguaje de programación por lo que podríamos encontrar pseudocódigos orientados a lenguajes de programación como C++, Phyton, Java, Pascal, etc. , así como también podemos encontrar pseudocódigos más generales o como una síntesis de varios lenguajes de programación.

El pseudocódigo no presenta un formato único por ser un lenguaje no formal, pero tampoco difiere demasiado de entre un programador u otro por lo tanto cualquier "experimentado" puede entender el pseudocódigo escrito en un estilo diferente por otra persona.

Mientras que los diagramas de flujo utilizan **las cajas**, el pseudocódigo hace uso de ciertas palabras que tienen un significado y que cumplen determinadas funciones, a estas se les denomina **Palabras Reservadas** o **Palabras Clave.**

***Ejemplo*:** Desarrolle un algoritmo para calcular el promedio de tres notas dadas como dato. Represente el algoritmo mediante Pseudocódigo.

```
Inicio
    Leer (Nota1)
    Leer (Nota2)
    Leer (Nota3)
    Promedio ← (Nota1+Nota2+Nota3) / 3
    Escribir (Promedio)
Fin
```

En este ejemplo las Palabras Reservadas son: **Inicio, Leer, Escribir** y **Fin.**

Se puede observar que el Pseudocódigo es una mezcla de lenguaje natural con algunas convenciones sintácticas que son propias de los lenguajes de programación.

1.4.2.1 VENTAJAS DEL USO DE PSEUDOCÓDIGO:

Entre las ventajas más importantes:

1. Permite una representación simple de operaciones muy complejas y repetitivas.
2. El espacio ocupado es relativamente corto a comparación con los diagramas de flujo.
3. Es fácil traducirlo a un lenguaje de programación por su similitud.

1.4.2.2 DESVENTAJA DEL USO DE PSEUDOCÓDIGO:

1. A diferencia de los diagramas de flujo, el pseudocódigo no ha sido estandarizado por lo que para su escritura no existen reglas rígidas estrictamente definidas.

OBSERVACIONES:

- ✓ En este curso debemos preocuparnos principalmente en dos cosas:
  1. Prestar la mayor atención en el análisis de los problemas y en la lógica de la solución.
  2. Aprender y adoptar un estilo de pseudocódigo para la representación de algoritmos.

- ✓ Continuamente durante el desarrollo de este curso será inevitable hacer analogías entre lo que es un Pseudocódigo y un Lenguaje de Programación para poder comprender el funcionamiento del primero en base al segundo.

- ✓ El pseudocódigo que presentamos en este texto está orientando al lenguaje de programación **Pascal**, así que la sintaxis de nuestro pseudocódigo será muy similar a este lenguaje, pero aun así esto no es ningún impedimento para posteriormente poder aprender cualquier lenguaje de programación estructurado.

2. CONCEPTOS BÁSICOS

A continuación se estudiará todo un conjunto de conceptos básicos pero muy importantes los cuales nos permitirán comprender la forma como se construye un programa. Detrás de todo programa existe un conglomerado de elementos que se encuentran interrelacionados y que permiten el funcionamiento del mismo. Los elementos básicos de programación son: Datos, tipos, identificadores, palabras reservadas, valores, constantes, variables, operadores, expresiones, instrucciones, estructuras, etc., estos se combinan siguiendo un conjunto determinado de reglas a lo que se le denomina **sintaxis**. La sintaxis del pseudocódigo no es única porque al no ser un lenguaje estándar, cada programador lo usa según su criterio. Los lenguajes de programación sí son estándar por lo tanto tienen sintaxis ya definidas.

2.1. DATO E INFORMACIÓN

No existe una definición única de **dato** e **información** ya que estos términos se encuentran presentes en todas las áreas del conocimiento. Se muestran a continuación algunas definiciones más cercanas a nuestro interés.

2.1.1. DATO:

- *Información dispuesta de manera adecuada para su **tratamiento por un ordenador**.*

- *El **dato** es una representación simbólica (numérica, alfabética, algorítmica etc.), un atributo o una característica de una entidad. Los datos son hechos que describen sucesos y entidades. No tienen ninguna información. Puede significar un **número**, una **letra**, o cualquier **símbolo** que representa una palabra, una cantidad, una medida o una descripción. El dato no tiene valor semántico (sentido) en sí mismo, pero si recibe un tratamiento (procesamiento) apropiado, se puede utilizar en la realización de cálculos o toma de decisiones. Es de empleo muy común en el **ámbito informático** y, en general, prácticamente en cualquier disciplina científica. **En programación**, un dato es la expresión general que describe las características de las entidades sobre las cuales opera un **algoritmo**.*

- *Es la expresión general que describe los objetos **con los cuales opera una computadora**.*

-*Es la **materia prima** que sirve de base para obtener información. También se dice que es un conjunto de caracteres o símbolos que no tiene un significado propio hasta que se le atribuye un significado asociándolo como parte de la descripción de alguna cualidad o característica perteneciente a algo. Los datos pueden ser normalmente discretos o continuos, pueden ser medidos o simplemente pueden describir una característica. En cualquier caso **sirven de base para su transformación en información**. En la mayoría de los casos, los datos ya existen y pueden pertenecer al entorno o al sistema, en otros casos los datos son el resultado de procesos previos y finalmente se tienen algunos datos que son inventados o definidos.*

2.1.2. INFORMACIÓN:

-Comunicación o adquisición de conocimientos que permiten ampliar o precisar los que se poseen sobre una materia determinada.

*- En sentido general, la **información** es un conjunto organizado de datos procesados, que constituyen un mensaje que cambia el estado de conocimiento del sujeto o sistema que recibe dicho mensaje.*

*- Es un recuso normalmente intangible que tiene la característica de que cuando se posee en cantidad, calidad y oportunidad adecuada, **ayuda a tomar mejores decisiones**. La información es el **resultado de la transformación de los datos**. La calidad de la información obtenida está relacionada con la calidad de los insumos o datos y con los conocimientos, técnicas o métodos que se elijan para efectuar la conversión.*

En conclusión:

Datos: Son pequeñas partes de información que por sí solas tal vez digan poco o nada pero ordenados de manera conjunta resultan útiles. Además tienen la cualidad de que pueden ser registrados por un computador.

Información: La información es un conjunto organizado de datos que se obtienen luego de haber pasado por un ordenamiento y procesamiento.

2.2. ATRIBUTOS DE UN DATO

Como bien sabemos, el presente curso nos permitirá aprender a diseñar algoritmos mediante la técnica de representación: **Pseudocódigo.** Esto es con el objetivo de tener el algoritmo ya listo y preparado para su posterior codificación en un Lenguaje de Programación. Todo programa de computadora trabaja continuamente con datos de entrada y mediante un conjunto de acciones o instrucciones devuelve datos de salida que constituyen la información deseada. Estos datos (entrada o salida) tienen tres características o atributos fundamentales:

- Tipo

Nos determina las operaciones que podemos realizar con el dato y la manera en que será codificada internamente en la computadora.

- Valor

Se refiere al contenido el cual puede cambiar o permanecer fijo.

- Identificador

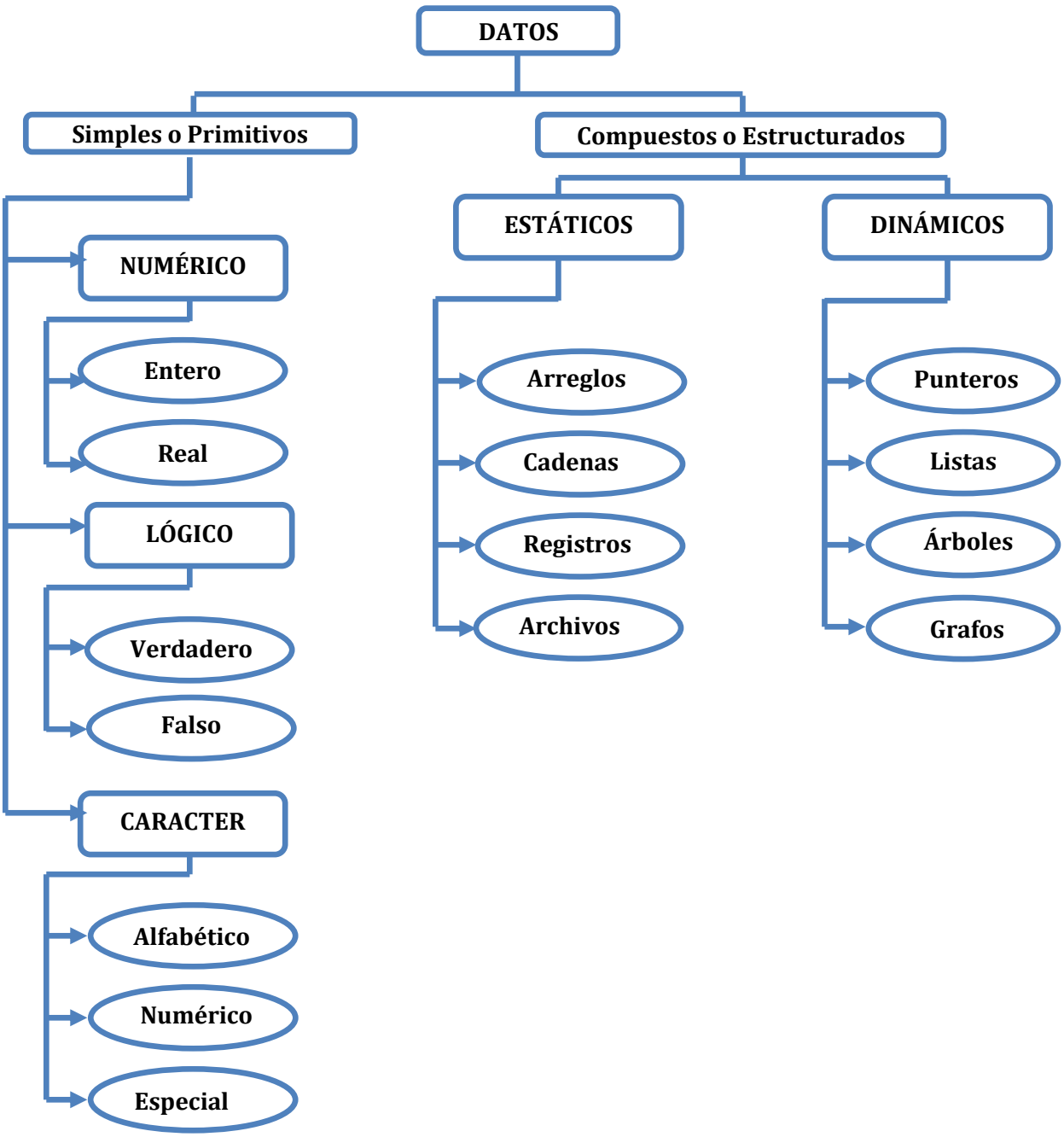
Es el nombre que permite referenciarlo y que además lo diferencia del resto.

2.2.1 PRIMER ATRIBUTO: TIPO

Desde el punto de vista de la programación los datos se clasifican en forma general de la siguiente manera:

2.2.1.1. DATOS SIMPLES: Reciben otros nombres equivalentes: Datos básicos, Datos primitivos, Datos estándar.

2.2.1.2. DATOS COMPUESTOS: Llamados también: Datos complejos o Datos estructurados.



DATOS SIMPLES

Vamos a clasificar a los datos simples según la **clase** o **tipo** de valores que puedan contener bajo dos enfoques:

A. Clasificación Principal

B. Clasificación en Ordinales y no Ordinales.

A. CLASIFICACIÓN PRINCIPAL: Los tipos de datos simples más importantes son:

- Datos de tipo Numérico
- Datos de tipo Carácter
- Datos de tipo Lógico.

1. NUMÉRICO: Estos datos nos van a permitir realizar las operaciones aritméticas comunes y otros cálculos matemáticos. Se dividen en dos: **enteros** y **reales**.

Numérico Entero: Son aquellos números enteros positivos y negativos que ya conocemos y por lo tanto no tienen parte decimal.

-2 , -1 , 0 , 1 , 2 , 458, 78989

Numérico Real: Todos aquellos que no son enteros serán considerados reales, por ende van a tener punto decimal y parte no entera. También son positivos y negativos.

-0.1, +8.33, -53.2 , etc.

2. CARACTER: También llamado **Alfanumérico**. Está conformado por dígitos, letras mayúsculas, minúsculas y símbolos especiales. Siempre van encerrados entre comillas simples (apóstrofos ‘ ’) y constan de un símbolo individual (carácter: una sola letra, un único dígito o un único símbolo). Se divide en tres principalmente: Alfabético, Numérico y Especial.

Carácter Alfabético: Corresponde a las letras del alfabeto ya sean mayúsculas o minúsculas.

‘A’ ‘B’ ‘C’ ‘D’ ‘E’ ‘F’ ‘G’ ‘H’ ‘I’ ‘J’ ‘K’ ‘L’ ‘M’ ‘N’ ‘O‘ ‘P‘ ‘Q‘ ‘R‘ ‘S’ ‘T’ ‘U’ ‘V’ ‘W’ ‘X’ ‘Y’ ‘Z’

‘a’ ‘b’ ‘c’ ‘d’ ‘e’ ‘f’ ‘g’ ‘h’ ‘i’ ‘j’ ‘k’ ‘l’ ‘m’ ‘n’ ‘o‘ ‘p‘ ‘q‘ ‘r‘ ‘s’ ‘t’ ‘u’ ‘v’ ‘w’ ‘x’ ‘y’ ‘z’

Carácter Numérico: No debe confundirlos con los datos ***numéricos*** anteriores. Un dato de tipo ***carácter numérico*** va encerrado entre comillas simples y existen solo diez datos de este tipo, además con estos no se pueden realizar operaciones aritméticas. Lo caracteres numéricos son:

‘1’ ‘2’ ‘3’ ‘4’ ‘5’ ‘6’ ‘7’ ‘8’ ‘9’ ‘0’

Aprecie que no es lo mismo el dato de tipo **numérico entero 3** que el dato de tipo **carácter numérico ‘ 3 ’**. El uso de los apóstrofos nos permitirá diferenciarlos.

Carácter Especial: Corresponden a ciertos símbolos especiales como los siguientes:

‘¿’ ‘¡’ ‘*’ ‘/’ ‘{‘ ‘@’ ‘%’ ‘Ñ’ ‘ñ’ ‘♀’ ‘◙’

3. LÓGICO: También llamado **Booleano**. Es un dato que solo puede tomar un valor de entre dos posibles: Verdadero y Falso. Son útiles al momento de realizar comparaciones entre datos además permiten evaluar condiciones. En este curso vamos a utilizar las letras **V** y **F** cuando hagamos referencia al valor **Verdadero** y **Falso** respectivamente.

B. SEGUNDA FORMA DE CLASIFICACIÓN: Datos de tipo Ordinal y no Ordinal.

1. ORDINALES: Vamos a considerar como ordinales a los siguientes datos:

- ✓ Los datos de tipo **numérico entero.**
- ✓ Los datos de tipo **carácter.** (Los caracteres se ordenan de acuerdo a su código ASCII).
- ✓ Los datos de tipo **lógico**: verdadero y falso.

La razón de la clasificación como ordinales es porque el conjunto de valores que asumen se pueden contar, es decir, podemos establecer una relación de uno a uno entre los valores de los datos y los números naturales.

2. NO ORDINALES: Vamos a considerar como no ordinales a los datos de tipo **numérico real**.

A diferencia de los de datos ordinales, los no ordinales no se pueden contar. No se puede establecer una relación de uno a uno entre ellos y los números naturales. Dicho de otra

forma, para que un dato se considere ordinal se tiene que poder determinar el anterior y el siguiente a este.

Ejemplo:

- Para el dato de tipo numérico entero: 5 , sabemos que antes está el 4 y luego está el 6.
- Dado el dato de tipo carácter alfabético: 'D' , sabemos que antes está 'C' y después está 'E'.
- Dado el dato de tipo carácter numérico: '5' , sabemos que antes está '4' y después está '6'.
- Dado el dato de tipo numérico real: 5.021 , no podemos determinar los reales que están antes o después, por lo tanto, es no ordinal.

Los datos lógicos también son ordinales pero solo pueden asumir uno de dos valores: **V** y **F** por lo que la relación de orden es el siguiente: **F < V**

OBSERVACIONES:

- ✓ En conclusión los **tipos** simples son: **entero**, **real**, **carácter** y **lógico**.
- ✓ De ahora en adelante en vez de mencionar la palabra completa: "***dato de tipo numérico entero***" solo se va a mencionar "***dato numérico entero***" o simplemente "***dato entero***". Lo mismo se aplica para el "*dato real*", "*dato carácter*" y "*dato lógico*".
- ✓ CÓDIGO ASCII: Los caracteres tienen un código que nos permite establecer un orden, de esta manera cuando comparamos caracteres, lo que en realidad se compara son sus códigos. La relación de orden entre los caracteres es el siguiente:

  ' 0 ' < 1 ' < ' 2 ' < ' 3 ' < ' 4 ' < ' 5 ' < ' 6 ' < ' 7 ' < ' 8 ' < ' 9 ' < ' A ' < ' B ' < ' C ' < ' D ' < ' E ' < ' F ' < ' G ' < ' H ' < ' I ' < ' J ' < ' K ' < ' L ' < ' M ' < ' N ' < ' O ' < ' P ' < ' Q ' < ' R ' < ' S ' < ' T ' < ' U ' < ' V ' < ' W ' < ' X ' < ' Y ' < ' Z ' < ' a ' < ' b ' < ' c ' < ' d ' < ' e ' < ' f ' < ' g ' < ' h ' < ' i ' < ' j ' < ' k ' < ' l ' < ' m ' < ' n ' < ' o ' < ' p ' < ' q ' < ' r ' < ' s ' < ' t ' < ' u ' < ' v ' < ' w ' < ' x ' < ' y ' < ' z '

- ✓ Se observa que el orden para los caracteres numéricos y alfabéticos es:

  Dígitos < mayúsculas < minúsculas.

- ✓ El carácter **espacio en blanco** es de tipo especial ' '
- ✓ A parte de los datos simples, existen datos llamados compuestos o estructurados. Los datos compuestos son agrupaciones de datos simples. La agrupación de datos simples se puede realizar de diferentes formas obteniéndose de esta manera distintos datos

compuestos como: Arreglos, Registros, etc. Por ahora solo nos es necesario conocer un tipo de dato **compuesto** que recibe el nombre de **Cadena de caracteres** o simplemente **Cadena.**

CADENA: Es un dato compuesto que resulta de la agrupación de varios datos de tipo carácter. Los datos de tipo cadena van entre comillas dobles.

Ejemplos:

"***Voy a aprobar el curso de algoritmos***"

"***Teodoro Córdova Neri***"

Los caracteres presentes en una cadena no son solo alfabéticos, sino pueden ser numéricos y hasta caracteres especiales.

"***Perú, Medalla de Bronce, Copa América 2011***"

"***theoliztik@gmail.com***"

Los datos de tipo cadena tienen la característica de tener una longitud. La longitud se refiere a la cantidad de caracteres que contiene, considerando además los caracteres de espacio en blanco.

Ejemplo:

"***Voy a aprobar el curso de algoritmos***"

Longitud: 36

2.2.2. SEGUNDO ATRIBUTO: VALOR

El valor es el contenido de un dato en un determinado momento. Dependiendo de si dicho valor puede cambiar o no durante el desarrollo del algoritmo (o ejecución del programa) estaremos hablando de **variables** y **constantes.**

- ✓ El conjunto de **valores** de los datos de tipo entero son números enteros positivos o negativos.
- ✓ El conjunto de **valores** de los datos de tipo carácter son letras, dígitos o símbolos individuales encerrados entre comillas simples.
- ✓ Un dato lógico solo puede tener como **valor**: Verdadero o Falso.
- ✓ Los datos reales tienen **valores** que serían números positivos o negativos con parte decimal.

2.2.3. TERCER ATRIBUTO: IDENTIFICADOR

Los identificadores son nombres o etiquetas que permiten ***identificar*** a los DATOS con los cuales trabajaremos y en general a todos ***los elementos de un algoritmo.*** Así como las

personas tenemos un nombre que nos identifica, en los algoritmos habrán componentes que necesitarán de un nombre o identificador único.

Los identificadores están constituidos por un conjunto de caracteres y son llamados ***válidos*** cuando poseen las siguientes características:

1. Todo identificador **debe comenzar con una letra** (no pueden empezar con un número o con otro símbolo) y no deben contener el carácter espacio en blanco ‘ ’

   Ejemplo:

   Son identificadores válidos:

   A1 , X1 , A1T3QM , A2B3C , AlgorRiTmo , b , cont

   Son identificadores no válidos:

   1AA , X 1

   (Los identificadores son no válidos porque el primero comienza con un número y el segundo contiene al carácter espacio en blanco)

2. Luego de la primera letra, los demás caracteres que forman el identificador pueden ser:
   - ✓ **caracteres** alfabéticos (letras mayúsculas o minúsculas)
   - ✓ caracteres **numéricos** (dígitos)
   - ✓ Únicamente al carácter ***Subguión*** o ***Guión bajo*** ‘ _ ’ que es usado para sustituir al espacio en blanco.

   Ejemplo:

   Son identificadores válidos:

   X_13 , The_Oliztik , Estructura_de_datos , Al_go_rit_mo

   Son identificadores no válidos:

   X-13 , The-Oliztik , 1Estructura---dedatos , @lgo®itmo

   (No confundir al carácter **guión bajo** ‘ _ ’ con el carácter **guión** ‘ - ’. El carácter guión no puede ser usado como parte de un identificador.)

3. Los identificadores deben de ser diferentes de las ***Palabras Reservadas***.

Las Palabras Reservadas son **Identificadores Predefinidos**. En cada lenguaje de programación existe un conjunto de palabras reservadas para determinados elementos del lenguaje, algunas de estas palabras realizan tareas específicas mientras que otras son solo etiquetas, en cualquiera de los casos, el identificador de algún dato no debe coincidir con las palabras reservadas ya existentes. En Pseudocódigo algunas de las palabras reservadas que existen y que usaremos a lo largo del curso son:

Inicio	**Cadena**	**Mientras**
Fin	**Arreglo**	**Hacer**
Leer	**Registro**	**fin_mientras**
Escribir	**mod**	**Desde**
entero	**div**	**fin_desde**
real	**Si**	**Repetir**
carácter	**Entonces**	**Hasta_que**
lógico	**sino**	**Verdadero**
Var	**fin_si**	**Falso**
Const	**En_Caso**	
Tipo	**fin_caso**	

NOTA:

- ✓ Cada lenguaje de programación tiene sus propias palabras reservadas las cuales se encuentran en inglés. Las palabras reservadas usadas en este texto serán en español.
- ✓ Cada lenguaje de programación tiene sus propias reglas en lo que respecta a los identificadores, pero en la mayoría de los casos coincide con las tres reglas descritas líneas arriba.
- ✓ Se recomienda que los identificadores sean significativos según el dato que representan.
- ✓ Cada dato debe tener un identificador único.

3. CONSTANTES Y VARIABLES

3.1 CONSTANTES

Son **datos** cuyo contenido no cambia por lo tanto van a tener un valor fijo. Las constantes son valores de un determinado tipo: entero, real, carácter o lógico por eso se clasifican en: Constantes enteras, constantes reales, constantes carácter y constantes lógicas. Existen también las constantes cadena.

A las constantes podemos darles un **identificador** o podemos optar por usarlas de forma explícita.

Ejemplos:

PI = 3.14	(Constante real)
N = 13	(Constante entera)
LETRA = 'A'	(Constante carácter)
VERDAD = V	(Constante lógica)
MENSAJE = "*Año* 2007"	(Constante cadena)

SUGERENCIA: Algunos autores recomiendan adquirir el hábito de identificar a una constante mediante caracteres en mayúscula.

3.2 VARIABLES

Son **datos** que no van a tener un valor fijo por lo que se va poder modificar su contenido. De forma análoga que las constantes, las variables almacenan valores de un determinado tipo por tanto existen variables enteras, variables reales, variables carácter, variables lógicas y variables cadena.

Las variables necesariamente deben tener un **identificador**.

Ejemplos:

Sueldo , edad , fecha , Ganancia, suma1 , suma_2 , i , n , cont , contA , cont_edad

La utilización de variables es vital porque permiten almacenar valores que serán procesados por la computadora para que finalmente sean entregados y resulten de mayor utilidad que en un inicio.

OBSERVACIONES:

- Las variables pueden cambiar únicamente su **Valor** (contenido) pero no su **Tipo** ni su I**dentificador.** Siempre deberán almacenar valores que pertenezcan al mismo tipo. Las constantes no pueden cambiar ninguno de sus tres atributos.
- Es importante saber distinguir los atributos de un dato. A continuación se presentan constantes en las que se indican sus tres atributos:

• PI = 3.14	Identificador: PI Valor: 3.14 Tipo: real
• N = 13	Identificador: N Valor: 13 Tipo: entero
• LETRA = ' A '	Identificador: LETRA Valor: 'A' Tipo: carácter
• VERDAD = V	Identificador: VERDAD Valor: V (Verdadero) Tipo: lógico
• MENSAJE = "*Año* 2007"	Identificador: MENSAJE Valor: "*Año* 2007" Tipo: Cadena

- No existe una regla fija para establecer el Identificador de una variable o constante por lo que queda a criterio de cada persona, eso sí, siempre se debe respetar las reglas básicas descritas inicialmente.

SUGERENCIA: Se recomienda que el identificador de una variable o constante tenga relación con los valores que almacena, de esta manera diseñaremos algoritmos o crearemos programas más fáciles de entender. Ejemplo: Si vamos a utilizar una variable que almacene edades, podríamos ponerle como identificador la palabra: ***edad***, aunque también podríamos ponerle otros nombres: ***x, qqq, L_o_qq*** pero el primer identificador es más significativo.

OBSERVACIONES:

En conclusión los datos tienen tres atributos: **Tipo**, **Valor** e **Identificador**. En programación, además de los tres atributos mencionados, se toma en cuenta un cuarto: el **Tamaño** ocupado en la memoria. Este tamaño se mide en bits o en bytes y depende del **Tipo** y del lenguaje de programación utilizado. En este curso no nos va a interesar este cuarto atributo.

3.3 LA VARIABLE COMO UNA POSICIÓN DE MEMORIA

Las variables son los elementos de programación que nos permitirán **almacenar** temporalmente diferentes valores durante el desarrollo del algoritmo (o ejecución del programa). Ahora bien. ¿En qué parte de una computadora se encuentran los datos (las variables y constantes) con los cuales trabajamos en un momento dado? Estos se localizan generalmente en la **memoria principal**.

3.3.1 MEMORIA

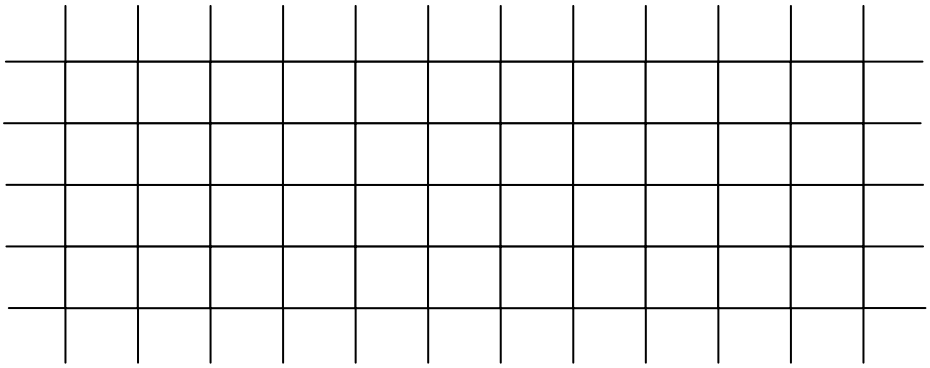
La memoria de una computadora se puede clasificar en dos partes: memoria principal y memoria secundaria. La memoria principal almacena los datos temporalmente (volátil) mientras que la memoria secundaria lo realiza de forma permanente.

Todo programa de computadora que aún no se ha *abierto*, es decir, que aún no se está ejecutando (vulgarmente: que aún no está corriendo), se encuentra almacenado en la memoria secundaria. (Disco Duro, USB, CD-ROM, etc.) En el momento que queramos usarlo, para que este pueda funcionar o **correr**, EL CPU (Microprocesador) se encarga de llevarlo desde la memoria secundaria hacia la memoria principal (Memoria RAM) mediante un proceso interno denominado "**carga"** el cual demora cierto tiempo. Una vez que el programa ha cargado y se encuentra funcionando, todos los datos que ingresemos (mientras el programa esté en ejecución) serán almacenados en la memoria principal. Cuando el programa finaliza o cuando se apaga la computadora (se corta la energía eléctrica), la información que estaba almacenada en la memoria principal desaparece pero la información de la memoria secundaria no. Por lo tanto, la memoria principal sirve para el almacenamiento de datos y programas que se estén usando en el momento actual.

La ventaja más importante que tiene la memoria principal es que es más rápida que la memoria secundaria (El tiempo de acceso es menor) y por lo tanto más costosa aunque su desventaja es que tiene menor capacidad de almacenamiento. Cuanto mayor es la memoria principal en una computadora, se podrá tener mayor cantidad de programas ejecutándose a la vez.

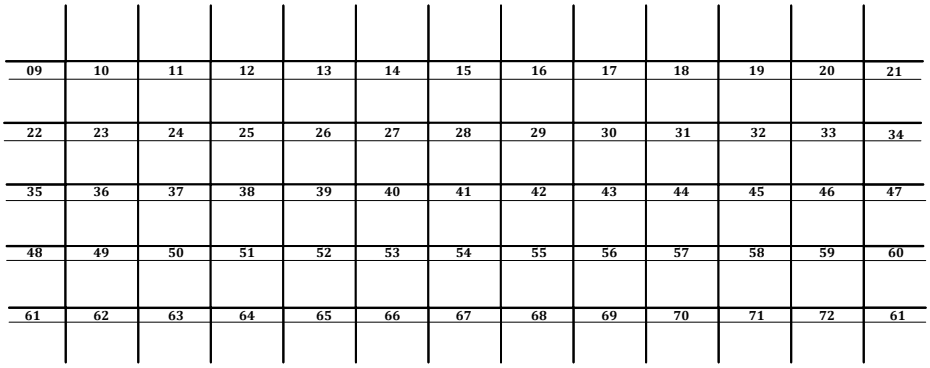
3.3.1.1 MEMORIA PRINCIPAL

La memoria principal está formada por millares de casilleros los cuales son una especie de *unidades de almacenamiento* llamados **celdas de memoria** o **posiciones de memoria.**



Cada uno de estos casilleros o celdas ocupa un lugar relativo, por lo tanto, tiene una ***dirección de memoria***. La dirección de memoria es un número único correspondiente a cada casillero que lo identifica y permite ubicarlo exactamente en la memoria.

Ejemplo: Se muestra la representación de un fragmento de la memoria principal. Los números que se encuentran en cada celda son sus respectivas **direcciones de memoria**. (Del 09 al 61)



Cada celda de memoria tiene la capacidad de almacenar datos e instrucciones.

¿De qué forma podemos almacenar un dato en la memoria?

Para poder almacenar algún dato en la memoria necesitamos en primer lugar **reservar** los casilleros o celdas necesarias para que alberguen al dato. La reservación de una parte de la memoria la realizaremos al momento de **declarar una variable.**

3.3.2 DECLARACIÓN DE UNA VARIABLE

La declaración de una variable consiste en especificar el **Tipo** y el **Identificador** para la variable. De esta forma nos veremos en la necesidad de declarar una variable por cada dato que vayamos a almacenar.

Sintaxis de la declaración de una variable:

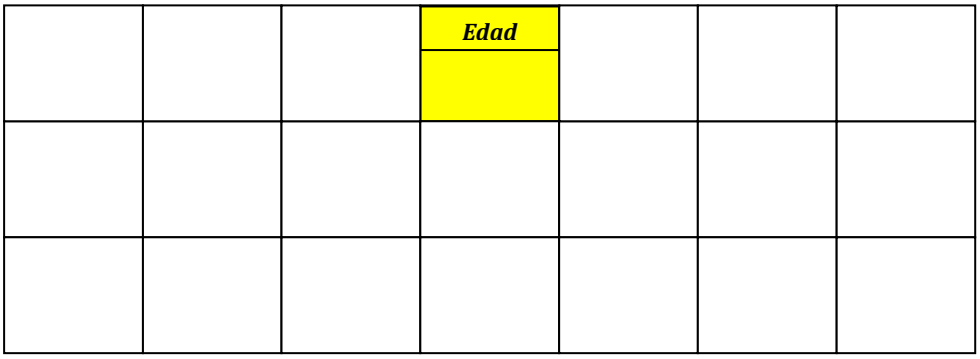
Var	*Identificador:* Tipo

- **Var:** Es una palabra reservada que indica que se está realizando la declaración de una o más variables.

Ejemplo: Necesitamos almacenar la Edad de una persona. Declare una variable que almacene valores enteros y que su identificador sea: ***Edad***

Var	*Edad*: entero

Al realizar esta declaración, internamente en la memoria principal se estaría reservando un casillero de la memoria para que allí se almacene un dato de tipo entero.

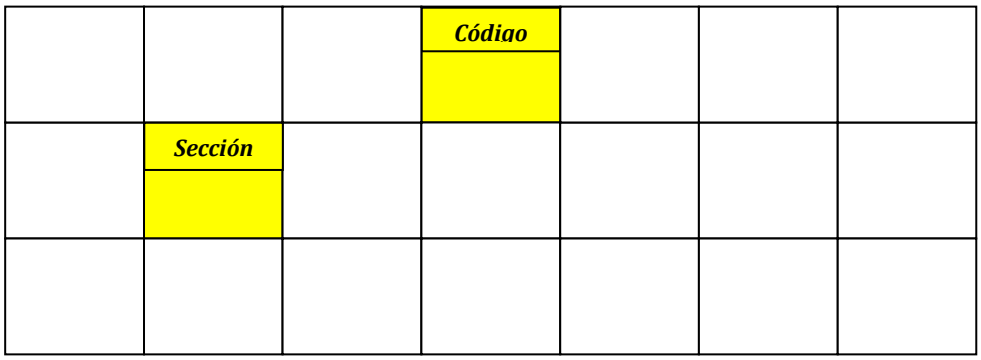


Por el momento la celda que ha sido reservada no almacena ningún valor.

Ejemplo: Declarar dos variables, una de tipo entero con identificador: ***Código***, y la otra de tipo carácter con identificador: ***Sección.***

Var	*Código*: entero
	Sección: carácter

Al realizar estas dos declaraciones, internamente en la memoria principal se estarían reservando dos casilleros de la memoria para que allí se almacene un dato entero y otro dato de tipo carácter.



La reservación de los casilleros es de forma aleatoria.

3.3.3. ACCESO A LA CELDA DE MEMORIA RESERVADA

La declaración nos permitió reservar las celdas necesarias para almacenar los datos que vayamos a usar. Pero ¿Cómo accedemos a dichas celdas? Estas celdas pueden encontrarse en cualquier parte de la memoria porque la reservación de estas es al azar. Es en este momento donde se observa la importancia de los **Identificadores** de variables. El identificador de una variable se comporta como la *dirección* de la celda reservada. De este modo, para almacenar un dato en una celda de memoria, debemos hacer referencia al Identificador de la variable que se utilizó al momento de la declaración.

Ejemplo: Almacene en la variable **Código** el valor numérico 13 y en la variable **Sección** el carácter 'A'. (Ambas variables ya fueron declaradas en el ejemplo anterior)

Para almacenar datos se usará mayormente dos instrucciones: instrucción de asignación e instrucción de lectura. Estas instrucciones serán estudiadas con más detalle más adelante, por ahora se presenta un ejemplo.

```
Código ← 13
Sección ← ' A '
```

			Código **13**			
	Sección **A**					

CONCLUSIONES:

Hemos conseguido notar que los datos en realidad son almacenados en las celdas de memoria. El almacenamiento de datos en la memoria es un proceso muy complicado por lo que en programación, se recurre al uso de variables lo que permite que este proceso sea más fácil de realizar:

Primero: Reservación de las celdas necesarias. Esto se realiza mediante la declaración de variables.

Segundo: Acceso a las celdas reservadas y almacenamiento. Esto se hace haciendo referencia al identificador de la variable declarada y utilizando ciertas instrucciones como la de asignación.

*El concepto de variable permite que el almacenamiento de datos sea más fácil de comprender y de realizar. Casi siempre en este texto y en otros se va a mencionar frases como la siguiente: "**La variable almacena un valor x**". Pero en realidad, lo que se quiere decir es que el valor se almacena en una celda de memoria y que para poder manipular este valor almacenado se utiliza el identificador de la variable como referencia a dicha celda. Es por esto que a una variable se le conoce también como **una posición de memoria**, porque se comporta como si fuera la celda en sí misma.*

OBSERVACIÓN

Las constantes al ser declaradas tienen también el mismo comportamiento que las variables, salvo una diferencia: Las constantes no pueden modificar su contenido. Una vez que se declaran, el contenido permanece invariable.

4. OPERADORES Y EXPRESIONES

OPERADORES

El **Tipo** de un dato (variable o constante) es el atributo que determina las operaciones que se pueden realizar con él. Para cada operación existen diferentes operadores. Los operadores se dividen en tres categorías:

- ✓ Operadores Aritméticos
- ✓ Operadores Relacionales
- ✓ Operadores Lógicos

Los datos que se encuentran afectados por los distintos operadores se denominan operandos.

EXPRESIONES

Una expresión está formada por la combinación de operandos y operadores.
Las expresiones también se clasifican en 3:

- ✓ Expresiones Aritméticas
- ✓ Expresiones Relacionales
- ✓ Expresiones Lógicas

Cuando se evalúa una expresión siempre se obtiene un **resultado**.

4.1 OPERADORES ARITMÉTICOS Y EXPRESIONES ARITMÉTICAS

4.1.1 OPERADORES ARITMÉTICOS

Los operadores aritméticos permiten la realización de cálculos aritméticos y básicamente son 6:

+, -, *, /, **div**, **mod.**

Ejemplo: 46 - 33

- ➢ Los números 46 y 33 son llamados **operandos** y el número que se obtiene al operar los operandos, se denomina **resultado.**
- ➢ Los operadores aritméticos son **binarios** porque afectan a los operandos agrupándolos de dos en dos de tal manera que se tenga un operando a la izquierda y otro a la derecha para que se lleve a cabo la operación.
- ➢ Los operadores **+** y **–** son binarios pero también se comportan como **unarios** o **monarios** porque al estar al lado izquierdo de un dato numérico (real o entero) le dan la característica de ser positivo o negativo.

4.1.2 EXPRESIONES ARITMÉTICAS

- Se denomina **Expresión Aritmética** a la combinación de **operandos numéricos** (datos constantes o datos variables) y **operadores aritméticos** (usando símbolos de separación si fuera necesario).

Ejemplo: a + (5 * 6 - edad)

- En una Expresión Aritmética los operandos son únicamente **datos numéricos** (entero y/o real) y siempre se obtiene un **resultado numérico** al evaluar dicha expresión.

TABLA DE OPERADORES ARITMÉTICOS

Operación	Operador	Tipos de operandos	Resultado	Ejemplo:
Adición	+	Entero + Entero	Entero	5 + 8
		Real + Real	Real	5.3 +7.7
		Entero + Real		5 + 7.9
Sustracción	-	Entero – Entero	Entero	20 – 7
		Real – Real	Real	20.5 – 7.5
		Entero - Reales		20 – 6.99
Multiplicación	*	Entero * Entero	Entero	13 * 1
		Real * Real	Real	26.0 * 0.5
		Entero * Real		26 * 0.5
División	/	Real / Real	**Real**	41.6 / 3.2
		Real / Entero		38.97 / 3
		Entero / Entero		
División Entera	***div***	Entero / Entero	Entero	26 **div** 2
Módulo (Residuo de la división entera)	***mod***	Entero *mod* Entero	Entero	1313 **mod** 100

EXPRESIONES ARITMÉTICAS EN FORMATO DE LÍNEA RECTA

Un formato de línea recta es aquel que no acepta la escritura en más de una línea. En algoritmos, todas las expresiones aritméticas deben de ser escritas en un **formato de línea recta**.

Ejemplo:

Formato Común	Formato de línea recta
$\frac{a+t+q}{13}$	(a + t + q) / 13
$a+\frac{t}{13}$	a + t/13
$a+t^2$	a + t * t
b^2-4ac	b * b – 4 * a * c
x^2+y^4	x * x + y * y * y * y

DIVISIÓN REAL Y ENTERA. OPERADORES div y mod

En aritmética existen dos tipos de división: división real y división entera.

Ejemplo:

División real	División entera
13 | 2 12 | 6.5 10 10 0 **Cociente**: 6.5 **Resto**: 0	13 | 2 12 | 6 1 **Cociente**: 6 **Resto**: 1

DIVISIÓN REAL

- Para realizar la división real se usará el operador **barra invertida /**. Este operador debe usarse en los siguientes casos:
  - Cuando ambos operandos sean reales. El resultado será real.
  - Cuando los operandos sean real y entero. El resultado será real.
  - Cuando ambos operandos sean enteros. El resultado será real.

  Conclusión: Con este operador, sin importar el tipo numérico (entero o real) al que pertenezcan sus operandos siempre se obtendrá un resultado real.

  Ejemplos:

  13.0 / 2.0 = 6.5 (Operandos reales, resultado real)
  13.0 / 2 = 6.5 (Operandos real y entero, resultado real)
  13 / 2.0 = 6.5 (Operandos entero y real, resultado real)
  13/2 = 6.5 (Operandos enteros, resultado real)
  26 / 2 = 13.0 (Operandos enteros, resultado real)

DIVISIÓN ENTERA

- Para realizar la división entera se usará el operador ***div***. Este operador debe usarse *únicamente cuando ambos operandos sean enteros* y siempre que se quiera obtener el cociente de la división entera.

  Ejemplo: Para la siguiente división entera:

  13 | 2
  12 | 6
  1

  En formato de línea recta sería: 13 ***div*** 2 . El uso del operador ***div*** nos da como resultado el número entero 6 que correspondería a ser el cociente de la división.

  13 ***div*** 2 = 6

Conclusión: Con este operador siempre se obtendrá un resultado entero.

- Para poder obtener el **residuo** o **resto de la división entera** se hará uso del operador ***mod.*** Este operador debe usarse únicamente cuando ambos operandos sean enteros.

  Ejemplo:

  13 ***mod*** 6 = 1

  Conclusión: Con este operador siempre se obtendrá un resultado entero.

  Ejemplos:

  52 ***div*** 4 = 13

  52 ***mod*** 4 = 0

  8 ***div*** 3 = 2

  8 ***mod*** 3 = 2

OBSERVACIONES:

- Para obtener el **resto de una división entera**, algunos autores utilizan el símbolo de porcentaje **%** en vez de ***mod***. En este texto usaremos únicamente ***mod.***
- Los operadores ***div*** y ***mod*** serán usados solo cuando los operandos sean enteros.
- El operador barra inclinada **/** permite realizar la división real y siempre se obtendrá un resultado real (los operandos pueden ser de cualquier tipo numérico).

  Ejemplo de uso incorrecto de operadores de división:

  La expresión 5.6 ***div*** 4 = ¿? no tiene sentido, pues 5.6 es real. Lo correcto sería usar el operador de barra inclinada 5.6 **/** 4 = 1.4

  La expresión 5.6 ***mod*** 4 = ¿? no tiene sentido, pues 5.6 es real y además no hay necesidad de conocer el residuo de una división real. Lo correcto sería 5 ***mod*** 4 = 1.

DISCREPANCIAS ENTRE AUTORES

- Algunos autores no utilizan el operador ***div*** para la división entera, sino que solo usan el operador barra inclinada **/** para la división sea esta real o entera. En ese caso, la regla es diferente, si ambos operandos son enteros, el resultado será entero y si por lo menos uno es real, el resultado saldrá real.

  Ejemplos:

  26/2 sería igual a 13 (entero) y no 13.0 (real) por ser ambos operandos enteros.

  26.0/2 sería igual a 13.0 (real) porque al menos un operando es real.

  26.0/2.0 sería igual a 13.0 (real) porque ambos operandos son reales.

26/2.0 sería igual a 13.0 (real) porque al menos un operando es real.

13/2 sería igual a 6 (entero) y no 6.5 (real) por ser ambos operandos enteros.

13.0/2 sería 6.5 (real) porque al menos un operando es real.

13.0/2.0 sería 6.5 (real) porque los operandos son reales.

13/2.0 sería 6.5 (real) porque al menos un operando es real.

NOTA IMPORTANTE:

No todos los lenguajes de programación tienen los mismos operadores aritméticos por lo tanto, nos veremos en la situación de hacer uso de más o de menos operadores según el lenguaje que elijamos cuando vayamos a programar. El pseudocódigo no se escapa de este pequeño problema, ya que al no ser estándar y al no existir una regla única para todos, cada autor usará los operadores que crea necesario según sus necesidades.

Ejemplo:

- En el lenguaje de programación conocido como **Pascal** se utilizan los operadores ***div*** y ***mod*** para la división entera y el operador barra inclinada **/** en la división real. Estos operadores usaremos también en este texto al escribir pseudocódigos.
- En el lenguaje de programación **C** y **C++** se utiliza el operador barra inclinada **/** para cualquier división.
- El operador de exponenciación ↑ existe en el lenguaje **BASIC**. En **FORTRAN** se utiliza **.

Estas diferencias en los lenguajes de programación no deben de preocuparnos por el momento, lo que debemos hacer es basar nuestro pseudocódigo en alguno de estos lenguajes (en nuestro caso Pascal) y aprenderlo bien. Luego ya habrá tiempo para estudiar y analizar las particularidades de cada Lenguaje.

ORDEN DE PRECEDENCIA

El orden de precedencia se refiere a la prioridad que tiene un operador con respecto a los otros. Cuando en una expresión aritmética existen más de dos operandos, el orden en el que se agrupan para su operación o evaluación se determina siguiendo un conjunto de reglas denominado **reglas de precedencia de operadores**, **reglas de jerarquía de operadores**, **reglas de prioridad**, etc.

NOTA: Los términos siguientes: precedencia, prioridad, jerarquía, superioridad, primacía, preferencia, preponderancia, predominio, etc., son sinónimos.

REGLAS DE PRECEDENCIA DE OPERADORES ARITMÉTICOS

Las reglas de precedencia de operadores aritméticos son las mismas que se aplican en la matemática ordinaria.

1. Los paréntesis tienen el mayor nivel de precedencia, por lo tanto, se evaluarán en primer lugar aquellas expresiones que se encuentran contenidas dentro de paréntesis.

   Ejemplo:

   (a + b) * c

   Se evalúa en primer lugar esta expresión

   Cuando una expresión contenga varios paréntesis en donde unos estén dentro de otros (a esto se le conoce como **paréntesis anidados**), las expresiones que se encuentren entre los dos paréntesis más internos serán evaluadas primero, hasta ir desapareciendo los paréntesis de par en par.

   Ejemplo:

   (13 ***mod*** ((7 ***div*** 3) * 2)) + 1

   (13 ***mod*** (2 * 2)) + 1

   (13 ***mod*** 4) + 1

   1 + 1

   2

2. Los operadores (*, /, ***div***, ***mod***) de las operaciones de **multiplicación**, **división** y **módulo** tienen el segundo mayor nivel de precedencia. Estos tres operadores tienen el mismo nivel de precedencia (con respecto a ellos mismos) por lo que si una expresión aritmética contiene varios, el orden de prioridad será de izquierda a derecha.

   Ejemplo:

   13 * 7 * 5

   Vemos que se repite el operador tres veces. Al tener el mismo nivel de precedencia, se **asociarán** los operandos en grupos de dos y de izquierda a derecha

   13 * 7 * 5

   91 * 5

   455

Ejemplo:

13 ***mod*** 7 * 5 ***div*** 2 / 2.0

Vemos que se encuentran presentes los operadores ***mod***, ******, ***div***, y ***/ ,*** como todos estos operadores tienen el mismo nivel de precedencia, se asociarán los operandos en grupos de dos y de izquierda a derecha.

13 ***mod*** 7 * 5 ***div*** 2 / 2.0
6 * 5 ***div*** 2 / 2.0
30 ***div*** 2 / 2.0
15 / 2.0
7.5

3. Los operadores (**+**, **-**) de las operaciones de Adición y Sustracción son los operadores aritméticos con el menor nivel de precedencia. Ambos tienen el mismo nivel de precedencia (entre ellos) por lo que si se encuentran presentes en una misma expresión, el orden de prioridad será de izquierda a derecha.

Ejemplo:

13 **+** 7 **-** 5 * 2

El operador de multiplicación tiene el mayor nivel de precedencia de entre todos los operadores presentes en la expresión aritmética actual. Luego, los operadores + y – tienen el mismo nivel de precedencia por lo que los operandos se asociarán de izquierda a derecha.

13 **+** 7 **-** 5 * 2
13 **+** 7 - 10
20 **-** 10
10

TABLA DE PRECEDENCIA DE OPERADORES ARITMÉTICOS

Operadores	Nivel de Precedencia
( )	Mayor ↓
*, /, ***div***, ***mod***	
+, -	Menor

4.2 OPERADORES RELACIONALES O DE COMPARACIÓN Y EXPRESIONES RELACIONALES

4.2.1 OPERADORES RELACIONALES

Los operadores relacionales permiten realizar comparaciones entre valores ***del mismo tipo*** (de la misma naturaleza) con la finalidad de conocer si se cumple o no alguna relación. Los operadores relacionales son los siguientes: > , < , >= , <= , = , <>

TABLA DE OPERADORES RELACIONALES

Relación	Operador
Igual	=
Diferente	<> o ≠
Mayor	>
Menor	<
Mayor o igual	>= o ≥
Menor o igual	<= o ≤

Ejemplo: 5 < 6

4.2.2 EXPRESIÓN RELACIONAL

Es aquella combinación de valores numéricos, datos o expresiones aritméticas y de un **operador relacional**. Las expresiones relacionales siempre retornan como resultado un valor lógico (**Verdadero o Falso**).

Ejemplos:

- Comparación de datos numéricos:

Expresión Relacional	Resultado
13 **<=** 7	Falso
13 **=** 13	Verdadero
16.0 **>** 8.5	Verdadero

- Comparación de Expresiones Aritméticas:

Expresión Relacional	Resultado
8 + 5 * 1 **<=** 5 + 2 * 0	Falso
13 ***mod*** 7 * 5 ***div*** 2 / 2.0 **=** 13 + 7 - 5 * 2	Falso

En primer lugar se evalúa cada expresión aritmética y luego se comparan los resultados obtenidos en cada expresión mediante el operador relacional.

- Comparación de datos lógicos:

Expresión Relacional	Resultado
F = V	Falso

En programación se considera que el valor lógico **Falso** es menor que el valor lógico **Verdadero.**

- Comparación de datos de tipo carácter:

Expresión Relacional	Resultado
'A' < 'B'	Verdadero
'a' = 'A'	Falso
'z' > 'a'	Verdadero
'1' = 'X'	Falso

Los caracteres tienen asociados códigos los cuales son números binarios (formados por 1 y 0) que permiten representarlos internamente, son estos códigos lo que las computadoras reconocen. De esta manera, existe una correspondencia entre los caracteres y sus respectivos códigos los cuales reciben el nombre de CÓDIGO ASCII. Cuando se realiza la comparación de caracteres, en realidad lo que se comparan son sus respectivos códigos.

Ejemplo: Algunos códigos ASCII

Código ASCII	Carácter
0100 0001	'A'
0100 0010	'B'
⋮	⋮
0101 1010	'Z'
0110 0001	'a'
0110 0010	'b'
⋮	⋮
0111 1010	'z'

Los códigos originalmente están en el sistema binario pero de esta forma son difíciles de recordar por lo que es más fácil expresarlos en el sistema decimal.

Código ASCII	Carácter
48	‘ 0 ’
49	‘ 1 ’
50	‘ 2 ’
51	‘ 3 ’
52	‘ 4 ’
53	‘ 5 ’
54	‘ 6 ’
55	‘ 7 ’
56	‘ 8 ’
57	‘ 9 ’
65	‘ A ’
66	‘ B ’
67	‘ C ’
68	‘ D ’
⋮	⋮
90	‘ Z ’
97	‘ a ’
98	‘ b ’
99	‘ c ’
100	‘ d ’
⋮	⋮
122	‘ z ’

En Resumen, la relación de orden entre los caracteres alfabéticos y numéricos queda establecida de la siguiente manera:

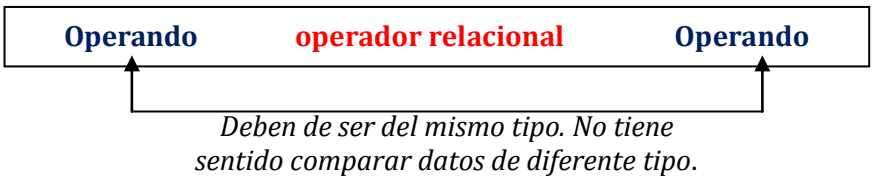
‘ 0 ‘ < 1 ’ < ‘ 2 ’< ‘ 3 ’< ‘ 4 ’< ‘ 5 ’< ‘ 6 ’< ‘ 7 ’< ‘ 8 ’< ‘ 9 ’ < ‘ A ’ < ‘ B ’ < ‘ C ’ < ‘ D ’ < ‘ E ’ < ‘ F ’ < ‘ G ’ < ‘ H ’ < ‘ I ’ < ‘ J ’ < ‘ K ’ < ‘ L ’ < ‘ M ’ < ‘ N ’ < ‘ O ‘< ‘ P ‘< ‘ Q ‘< ‘ R ‘< ‘ S ’ < ‘ T ’ < ‘ U ’ < ‘ V ’ < ‘ W ’ < ‘ X ’ < ‘ Y ’ < ‘ Z ’ < ‘ a ’ < ‘ b ’ < ‘ c ’ < ‘ d ’ < ‘ e ’ < ‘ f ’ < ‘ g ’ < ‘ h ’ < ‘ i ’ < ‘ j ’ < ‘ k ’ < ‘ l ’ < ‘ m ’ < ‘ n ’ < ‘ o ‘< ‘ p ‘< ‘ q ‘< ‘ r ‘< ‘ s ’ < ‘ t ’ < ‘ u ’ < ‘ v ’ < ‘ w ’ < ‘ x ’ < ‘ y ’ < ‘ z ’

- Comparación de datos de tipo cadena:

Expresión Relacional	Resultado
“ Oliztik ” = “ oliztik ”	Falso
“ A ” ≠ ” B “	Verdadero

La comparación de datos cadena se realiza carácter a carácter.

En general las Expresiones Relacionales tendrán la siguiente sintaxis:



Como se observa, las Expresiones Relacionales siempre constan de un **operador relacional** y dos **operandos** que no necesariamente son siempre numéricos, sino también pueden ser caracteres, cadenas, expresiones aritméticas, datos lógicos, variables, etc., pero siempre deben ser del mismo tipo. Como resultado de la evaluación de una Expresión Relacional, se obtiene siempre un dato de tipo lógico.

TABLA DE PRECEDENCIA DE OPERADORES RELACIONALES

Operadores	Nivel de Precedencia
<, >, <=, >=,	Mayor ↓
= , ≠	Menor

Ejemplo:

4 <= 2 = 8 > 10

Los operadores <= y > tienen la mayor precedencia, por lo tanto se comparan primero los operandos afectados por dichos operadores.

4 <= 2 = 8 > 10

F = F

V

4.3 OPERADORES LÓGICOS Y EXPRESIONES LÓGICAS

4.3.1 OPERADORES LÓGICOS

Los operadores lógicos son aquellos símbolos que conocemos en Lógica con el nombre de **conectivos lógicos** y son equivalentes a las conjunciones gramaticales y, o, y al adverbio de negación **no**. Los operadores lógicos van a permitir enlazar dos o más variables lógicas (lo que se conoce en Lógica como variables proposicionales) o dos expresiones (que den como resultado un valor lógico) y como resultado se obtendrá un valor lógico (verdadero o falso), esto se determina con la **tabla de verdad** correspondiente a cada operador. Los operadores lógicos son: ***y*** , ***o*** , ***no***

TABLA DE OPERADORES LÓGICOS

Relación	Operador
no	¬ ó ~
y	∧
o	∨

TABLAS DE VERDAD

NEGACIÓN ~

Valor Lógico	¬
V	F
F	V

El operador de negación es un operador **Unario** o **Monario** pues afecta a un único operando.

Sintaxis:

~ Operando

Donde: Operando puede ser: - Variable Lógica
- Constante Lógica
- Expresión Relacional
- Expresión Lógica

CONJUNCIÓN ∧

Valor Lógico		∧
V	V	V
V	F	F
F	V	F
F	F	F

DISYUNCIÓN ∨

Valor Lógico		∨
V	V	V
V	F	V
F	V	V
F	F	F

Los operadores de conjunción y disyunción son operadores **Binarios** pues involucran dos operandos.

Sintaxis:

Operando1 ∧ Operando2

Operando1 ∨ Operando2

Donde: Operando puede ser:
- Variable Lógica
- Constante Lógica
- Expresión Relacional
- Expresión Lógica

4.3.2 EXPRESIÓN LÓGICA

Es aquella combinación de variables lógicas, Expresiones Relacionales y de **operadores lógicos.** Las Expresiones Lógicas siempre retornan como resultado un valor lógico (**Verdadero o Falso**) el cual se determina con ayuda de las tablas de verdad.

Ejemplo:

~ (13 ***mod*** 7 * 5 ***div*** 2 / 2.0 **=** 13 **+** 7 **-** 5 * 2)

Operador Lógico — *Expresión Relacional*

Al evaluar la Expresión Relacional se obtiene como resultado el valor lógico **Falso.**

~ F

Según la tabla de verdad correspondiente al operador lógico de negación, el resultado final sería **Verdadero**.

Ejemplo:

" Oliztik " = " oliztik " ∧ 8 + 5 * 1 <= 5 + 2 * 0

Expresión Relacional — *Operador Lógico* — *Expresión Relacional*

Al evaluar cada Expresión Relacional se obtienen como resultados dos valores lógicos, cada uno a ambos lados del operador lógico:

$$F \wedge F$$

Según la tabla de verdad correspondiente al operador lógico de conjunción, el resultado final sería **Falso**.

Podemos decir que las Expresiones Lógicas son una extensión de las Expresiones Relacionales porque permiten unir en una sola expresión muchas relaciones a la vez.

Ejemplo:

$$((\underbrace{5 < 6}_{\text{Expresión Relacional}} \wedge \underbrace{6 < 10}_{\text{Expresión Relacional}}) \vee \underbrace{(10 = 9)}_{\text{Expresión Relacional}}) \wedge \sim \underbrace{(\text{“C++ ” = “ c++ ”})}_{\text{Expresión Relacional}}$$

Expresión Lógica (5 < 6 ∧ 6 < 10) — *Expresión Lógica* (10 = 9) — *Expresión Lógica* (“C++ ” = “ c++ ”)

Expresión Lógica

La evaluación sería de la siguiente manera:

$$((\mathbf{V} \wedge \mathbf{V}) \vee \mathbf{F}) \wedge \sim \mathbf{F}$$

$$(\mathbf{V} \vee \mathbf{F}) \wedge \mathbf{V}$$

$$\mathbf{V} \wedge \mathbf{V}$$

$$\mathbf{V}$$

TABLA DE PRECEDENCIA DE OPERADORES LÓGICOS

Operadores	Nivel de Precedencia
~	Mayor ↓ Menor
∧	
∨	

Ejemplo:

$$(5 < 6) \wedge (6 < 10 \quad \vee (10 = 9) \wedge \sim (\text{“C++ ” = “ c++ ”})$$

En primer lugar debemos evaluar las expresiones relacionales que se encuentran entre paréntesis. (Por regla general, los paréntesis tienen la mayor precedencia frente a cualquier operador)

$$\underbrace{(5 < 6)}_{\mathbf{V}} \wedge \underbrace{(6 < 10)}_{\mathbf{V}} \vee \underbrace{(10 = 9)}_{\mathbf{F}} \wedge \sim \underbrace{(\text{“C++ ” = “ c++ ”})}_{\mathbf{F}}$$

$$\mathbf{V} \wedge \mathbf{V} \vee \mathbf{F} \wedge \sim \mathbf{F}$$

En segundo lugar, el operador de negación tiene el mayor nivel de precedencia.

V ∧ V ∨ F ∧ ~ F

V ∧ V ∨ F ∧ V

En la expresión resultante, el operador de conjunción tiene la prioridad.

V ∧ V ∨ F ∧ V

V ∨ F

V

TABLA DE PRECEDENCIA TOTAL

Operadores	Nivel de Precedencia
( )	Mayor
~	
*, /, **div**, **mod**	
+, -	
<, >, <=, >=,	
= , ≠	
∧	
∨	Menor

Esta tabla permite evaluar correctamente aquellas expresiones en las que aparezcan operadores aritméticos, relacionales y lógicos.

Ejemplo:

5 < 6*1 ∧ 6 < 10 ∨ 10 = 9 ∧ ~ (“C++ ” = “ c++ ”)

En primer lugar debemos evaluar la expresión relacional que se encuentran entre paréntesis.

5 < 6*1 ∧ 6 < 10 ∨ 10 = 9 ∧ ~ (“C++ ” = “ c++ ”)

5 < 6*1 ∧ 6 < 10 ∨ 10 = 9 ∧ ~ F

En segundo lugar, el operador de negación tiene la prioridad entre los operadores restantes.

5 < 6*1 ∧ 6 < 10 ∨ 10 = 9 ∧ ~ F

5 < 6*1 ∧ 6 < 10 ∨ 10 = 9 ∧ V

La prioridad lo tiene ahora el operador aritmético de multiplicación.

5 < 6*1 ∧ 6 < 10 ∨ 10 = 9 ∧ V

5 < 6 ∧ 6 < 10 ∨ 10 = 9 ∧ V

Ahora la prioridad lo tienen los operadores relacionales de desigualdad.

5 < 6 ∧ 6 < 10 ∨ 10 = 9 ∧ V

V ∧ V ∨ 10 = 9 ∧ V

A continuación la prioridad es del operador relacional de igualdad.

V ∧ V ∨ 10 = 9 ∧ V
V ∧ V ∨ F ∧ V

La prioridad lo tiene el operador lógico de conjunción y por último el de disyunción.

V ∧ V ∨ F ∧ V
V ∨ F
V

OBSERVACIONES:

- Cada lenguaje de programación tiene sus propias reglas de precedencia de operadores, estas reglas propias de cada lenguaje presentan ligeras variaciones. En el caso que no tengamos en claro la prioridad de un operador sobre otro, podemos recurrir al uso de paréntesis para forzar el orden de evaluación ya que estos siempre tendrán la mayor precedencia.

  Ejemplo: Si no recordamos la tabla de precedencia de operadores, podemos anidar los paréntesis para priorizar aquellas operaciones a realizar primero.

((5 < (6*1)) ∧ (6 < 10)) ∨ ((10 = 9) ∧ (~ ("C++ " = " c++ ")))
((5 < (6*1)) ∧ (6 < 10)) ∨ ((10 = 9) ∧ (~ ("C++ " = " c++ ")))
((5 < (6*1)) ∧ (6 < 10)) ∨ ((10 = 9) ∧ (~ F))
((5 < (6*1)) ∧ (6 < 10)) ∨ ((10 = 9) ∧ V)

((5 < (6*1)) ∧ (6 < 10)) ∨ ((10 = 9) ∧ V)
((5 < 6) ∧ (6 < 10)) ∨ ((10 = 9) ∧ V)

Ahora la prioridad lo tienen los operadores relacionales de desigualdad.

((5 < 6) ∧ (6 < 10)) ∨ ((10 = 9) ∧ V)
(V ∧ V) ∨ ((10 = 9) ∧ V)

A continuación la prioridad es del operador relacional de igualdad.

(V ∧ V) ∨ ((10 = 9) ∧ V)
(V ∧ V) ∨ (F ∧ V)

La prioridad lo tiene el operador lógico de conjunción y por último el de disyunción.

(V ∧ V) ∨ (F ∧ V)
V ∨ F
V

Las Expresiones Lógicas y Relacionales se encontrarán presentes únicamente al inicio de las estructuras de control selectivas, fuera de estas no tienen ninguna utilidad. Lo relacionado a estructuras de control será estudiado más adelante.

CONCLUSIONES

- **Expresión Aritmética:**
  - **Operandos:** De tipo numérico.
  - **Operadores:** Aritméticos: **+**, **-**, ***, **/**, ***div*, *mod***
  - **Resultado:** De tipo numérico.
- **Expresión Relacional:**
  - **Operandos:** Expresiones Aritméticas, Datos del mismo tipo.
  - **Operadores:** Relacionales: =, <>, >, <, <=, >=
  - **Resultado:** De tipo lógico (Verdadero o Falso)
- **Expresión Lógica:**
  - **Operandos:** Expresiones Relacionales, Datos de tipo Lógico.
  - **Operadores:** Lógicos: ~ , ∧ , ∨
  - **Resultado:** De tipo lógico (Verdadero o Falso)

5. INSTRUCCIONES

5.1 INSTRUCCIÓN DE ASIGNACIÓN

La instrucción de asignación es la operación que permite almacenar valores en una variable. Para realizar la operación de asignación se utiliza el **operador** ← (flecha apuntando a la izquierda).

Sintaxis de la instrucción de asignación:

variable ← valor

Donde:

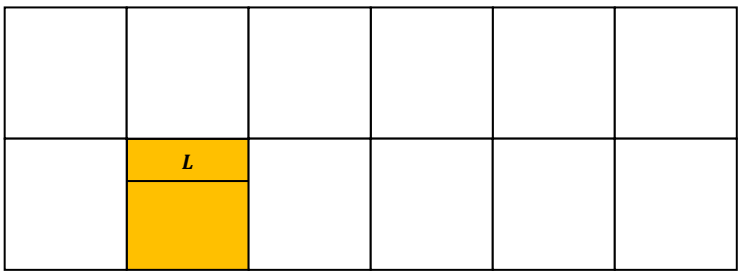
Variable: viene a ser el identificador de la variable.

Valor: es el valor que se le quiera asignar o almacenar.

Ejemplo: Declaremos una variable con los siguientes atributos: **Tipo**: Entero, **Identificador**: L
La declaración sería de la siguiente manera:

Var *L*: entero

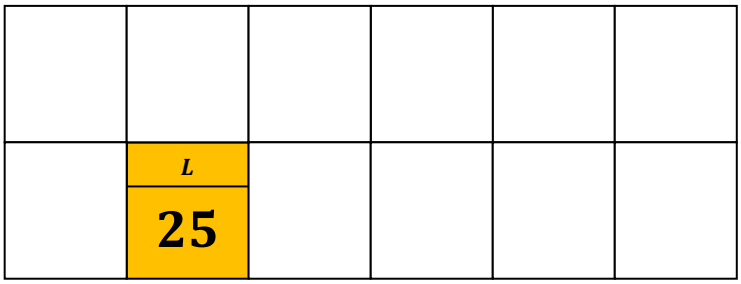
Internamente en la memoria central se reservaría un espacio de la memoria como se muestra en la figura siguiente:



Usemos ahora la instrucción de asignación para almacenar valores a la variable. Para esto, hacemos referencia al **Identificador** el cual se comporta como si fuera la dirección de la celda de memoria en donde se va a almacenar el valor.

L ← 25

Significa que a la variable de identificador L se le ha asignado el número entero 25. Por tal motivo el valor que almacena la variable en este momento es 25.

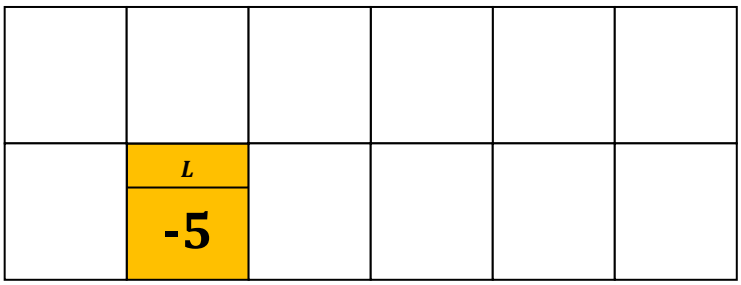


L ← 40

Al realizar esta asignación estamos haciendo que la variable cuyo identificador es L almacene al número entero 40.

L ← 5

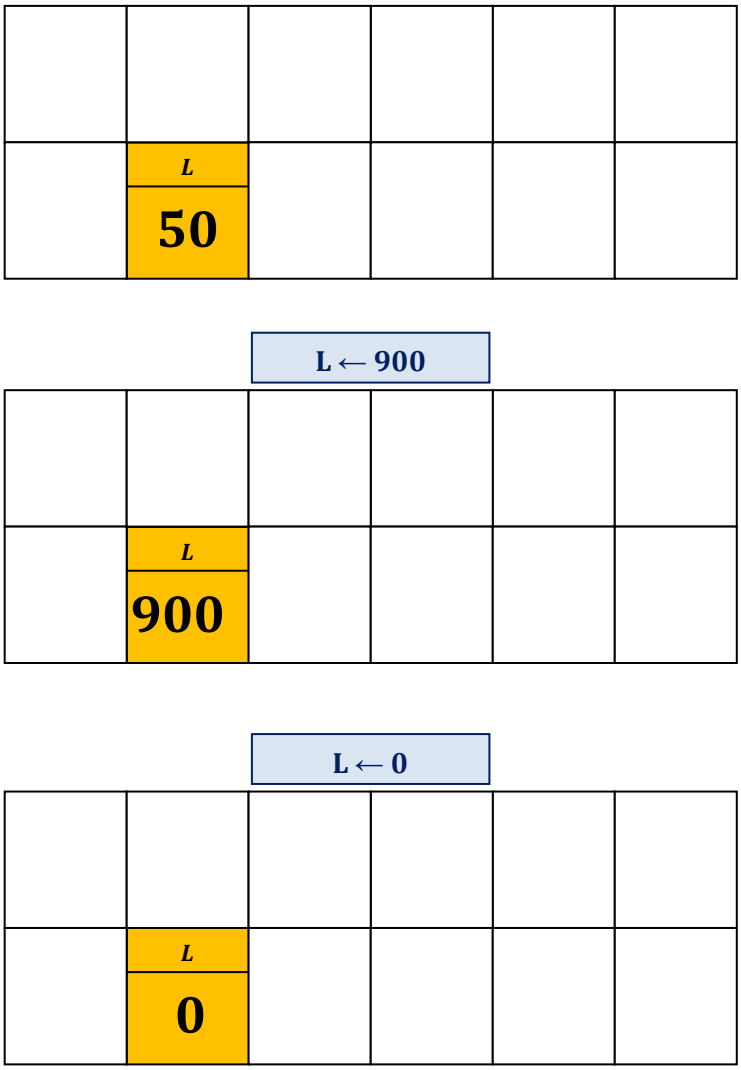
Al realizar esta tercera asignación, estamos haciendo que la variable cuyo identificador es L tenga como valor al número entero -5.



Como vemos, hemos realizado tres asignaciones consecutivas a la misma variable. De los tres valores que le hemos asignado, el último valor que almacenó la variable cuyo identificador es L es -5. ¿Qué sucedió con los valores: 25 y 40? Lo que sucedió fue que esos valores se perdieron o desaparecieron.

Ahora volvamos a realizar otras tres asignaciones a la misma variable en forma consecutiva:

L ← 50



El valor último que toma la variable de identificador L es el número entero 0. Los valores 50 y 900 desaparecieron.

La instrucción de asignación tiene la cualidad de ser ***destructiva*** porque elimina cualquier valor que haya estado almacenado anteriormente en una variable.

5.1.1. Asignación de una expresión

La instrucción de asignación también se puede realizar cuando al lado derecho del operador de asignación se encuentre una expresión (Aritmética, Lógica, Relacional). En este caso, en primer lugar se evalúa la expresión de la derecha y seguidamente, el valor obtenido como resultado de la evaluación es el que se asigna a la variable que se encuentra a la izquierda.

Sintaxis:

Variable ← expresión

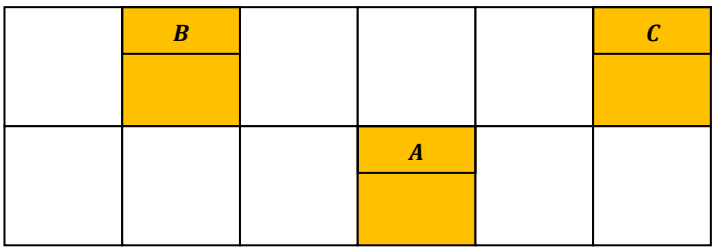
Ejemplo:

```
L ← 5 + 10
```

Significa que el valor que toma la variable (cuyo identificador es) L sería el resultado de sumar 5 + 10, es decir, el número entero 15.

Ejemplo: Declaremos tres variables enteras cuyos identificadores son: A, B, C.

```
Var    A, B, C: entero
```




```
A ← 10
```

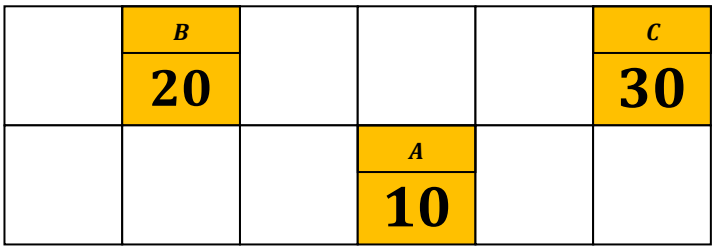
A la variable (cuyo identificador es) A se le asigna el número entero 10

```
B ← 20
```

A la variable (cuyo identificador es) B se le asigna el número entero 20

```
C ← A +B
```

A la variable (cuyo identificador es) C se le asigna el resultado de sumar los valores de las variables A y B, el cual sería 10 + 20 = 30.



IMPORTANTE: *Las computadoras realizan la asignación de una expresión en dos pasos.* ***Primero****: Evalúan la expresión del lado derecho y obtienen un resultado.* ***Segundo****, almacenan el resultado en la variable que se encuentra a la izquierda.*

Ejemplo: Declararemos una variable entera con identificador F.

Var *F*: entero

F ← 35

Ahora vamos a utilizar el mismo identificador en ambos lados del operador de asignación de la siguiente manera:

F ← F+10

Aunque no lo parezca, la asignación **F ← F+10** es coherente. El proceso de asignación se realiza en el orden siguiente:

Primero: Se calcula el resultado de la derecha, es decir la suma del valor actual de F (el cual es 35) más la constante 10.

Segundo: el resultado (45) se le asigna a la variable F que se encuentra a la izquierda del operador de asignación, por lo que este se convierte en el valor actual de F. Desde el punto de vista matemático esto no tendría sentido, pero sí en algoritmos y programación.

OBSERVACIONES:

- ✓ Resulta más cómodo referirse al identificador de la variable como la variable en sí misma. **Ejemplo:** Siendo estrictos, lo correcto sería mencionar: "*la variable cuyo identificador es x*", pero resulta más cómodo decir simplemente: "*la variable x*".
- ✓ Tener presente que la **variable** debe de ser del mismo tipo que el **valor** que le asignamos. Si una variable ha sido declarada de tipo entero, a esta solo se le puede asignar números enteros. Lo mismo ocurre para variables de otros tipos.

Ejemplo: Declaración de una variable de tipo carácter.

La declaración sería:

Var	*letra* : carácter

letra ← ' n '	Esta asignación es correcta porque **' n '** es un carácter.
letra ← 900	Esta asignación es incorrecta porque **900** es un entero.
letra ← 5.6	Esta asignación es incorrecta porque **5.6** es un real.
letra ← F	Esta asignación es incorrecta porque **F** es un dato lógico.

5.1.2 ERROR DE TIPO

Se comete un **error de tipo** cuando se le asigna valores de un tipo a una variable de otro tipo de datos. En el ejemplo anterior, en las tres últimas asignaciones se está cometiendo errores de tipo.

No tiene sentido usar asignaciones a una constante porque estas no pueden modificar su valor.

Ejemplo: Se realiza la siguiente declaración:

Var	*A, B, C, cuenta* : entero *Prom*: real

Con esta declaración se logra reservar cuatro espacios en la memoria. (Los espacios se reservan de forma aleatoria)

C		*B*			
	A		*cuenta*		*Prom*

Se realiza las siguientes asignaciones:

cuenta ← 0
A ← 13

C		*B*			
	A		*cuenta*		*Prom*
	13		**0**		

cuenta ← *cuenta* + 1
B ← 8

C		*B*			
	A	**8**	*cuenta*		*Prom*
	13		**1**		

cuenta ← *cuenta* + 1
C ← 13 - 8

C		*B*			
5		**8**			
	A		*cuenta*		*Prom*
	13		**2**		

$cuenta \leftarrow cuenta + 1$
$Prom \leftarrow (A + B + C) / 3$

C		*B*			
5		**8**			
	A		*cuenta*		*Prom*
	13		**3**		**6.5**

5.2. INSTRUCCIONES DE ENTRADA Y SALIDA

5.2.1. INSTRUCCIÓN DE LECTURA

La instrucción de lectura o también conocida como **entrada de datos** permite **leer** valores que son ingresados a la computadora para luego asignarlos a determinadas variables, en otras palabras, permite el ingreso de datos desde el exterior. De los dispositivos que permiten introducir datos a la computadora se encuentran entre los más usuales al teclado, mouse, cámara web, escáner (imágenes) y el micrófono (sonido) siendo el más común el teclado.
En el presente curso vamos a considerar exclusivamente al **teclado** como dispositivo de entrada estándar.

Sintaxis para la instrucción de lectura:

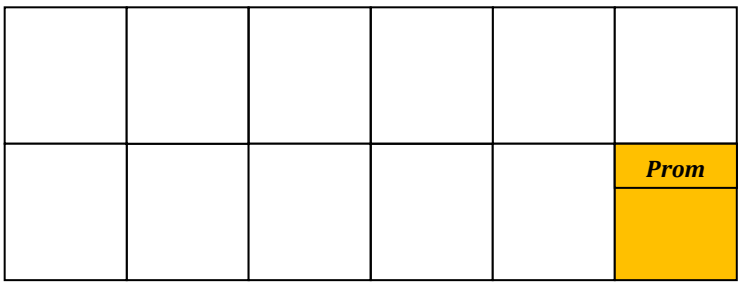
Leer (*variable*)

Cuando la computadora ejecuta la instrucción de lectura, espera a que el usuario introduzca un valor para la variable, por lo tanto, el usuario deberá responder escribiendo por el teclado algún valor (el cual debe de corresponder con el tipo de la variable para que no se cometa error de tipo) y seguidamente oprimirá la tecla *enter* para enviar los datos ingresados a la computadora. (Los datos que ingresa son visualizados en la pantalla).

Ejemplo: Se declara una variable de tipo real.

Var *Prom*: real

Con la declaración se logra reservar un espacio de la memoria.

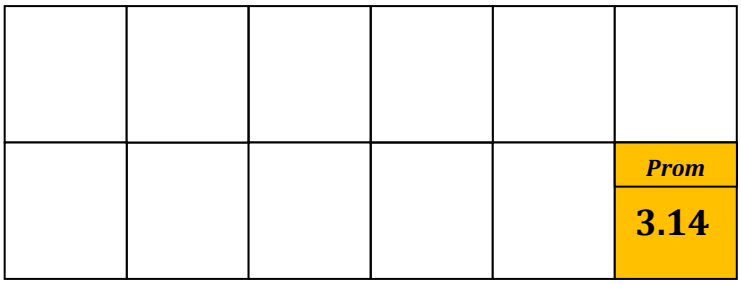


Luego se utiliza la siguiente instrucción de lectura:

Leer (*Prom*)

Con la instrucción anterior, se le está ordenando a la computadora que espere a que el usuario (que esté haciendo uso de la computadora) ingrese un valor real para que sea almacenado en la variable *Prom*. Si por ejemplo, un usuario escribe por el teclado el número **3.14** y seguidamente

presiona la tecla **enter**, inmediatamente este valor ingresa a la computadora y es asignado a la variable.



Para el caso en el que se necesiten ingresar varios datos, se necesitará utilizar muchas variables, por lo que la lectura se puede realizar de dos maneras:

* Utilizando una instrucción por cada variable:

```
Leer (variable_1)
Leer (variable_2)
      .
      .
      .
Leer (variable_N)
```

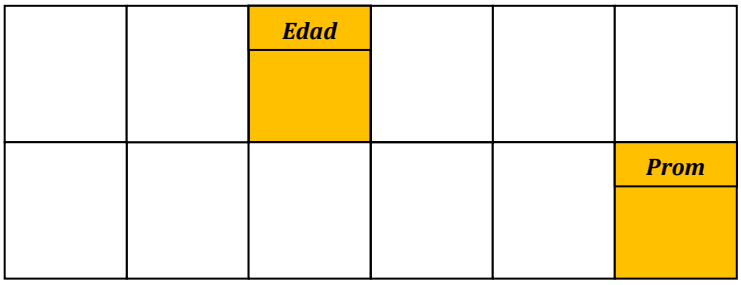
***** Utilizando una única instrucción de lectura para todas las variables:

```
Leer (variable_1, variable_2, ..... , variable_N)
```

Ejemplo: Se declaran dos variables, una de tipo real y la otra de tipo entero.

```
Var    Prom: real
       Edad: entero
```

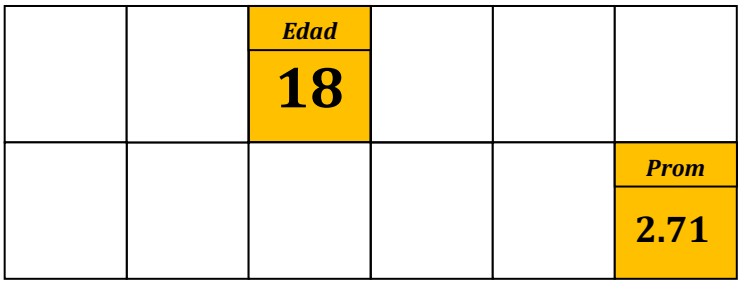
Con la declaración se logran reservar dos espacios de la memoria.



Luego se escriben las siguientes instrucciones de lectura:

```
Leer (Edad)
Leer (Prom)
```

Con las instrucciones anteriores, se le está ordenando a la computadora que espere a que el usuario (que esté haciendo uso de la computadora) ingrese primero un valor entero para que sea almacenado en la variable *Edad*, y seguidamente un valor real para que lo almacene en la variable *Prom*. Si por ejemplo, un usuario escribe por el teclado el número **18** y enseguida presiona la tecla **enter**, inmediatamente este valor ingresa a la computadora y es asignado a la variable *Edad*, luego si ingresa **2.71** y enseguida presiona enter, este valor es almacenado en la variable *Prom*.



NOTA: En lugar de utilizar dos instrucciones de Lectura, se pudo usar solo una en donde las variables quedan separadas por una coma.

```
Leer (Edad, Prom)
```

Ejemplo: Recordemos el ejemplo en el que desarrollamos el algoritmo para calcular el promedio de tres notas:

```
Inicio
    Leer (Nota_1)
    Leer (Nota_2)
    Leer (Nota_3)
    Promedio ←Nota_1+Nota_2+Nota_3) / 3
    Escribir (Promedio)
Fin
```

En este algoritmo (representado por medio de pseudocódigo) se puede observar que se están realizando tres instrucciones de lectura para cada una de las variables.

❶ **Leer** (*Nota*_1) Al utilizar esta instrucción se le está informando a la computadora que se va a ingresar un valor por medio del teclado y (luego de presionar enter) el valor ingresado deberá ser enviado para ser asignado a la variable *Nota*_1 .

❷ **Leer** (*Nota*_2) Al utilizar esta instrucción se le está informando a la computadora que se debe ingresar un valor por medio del teclado y que seguidamente este valor ingresado debe de ser asignado a la variable *Nota*_2 .

❸ **Leer** (*Nota*_3) Al utilizar esta instrucción se le está informando a la computadora que se va a ingresar un valor por medio del teclado y que seguidamente este valor ingresado debe de ser asignado a la variable *Nota*_3 .

El algoritmo también se pudo haber diseñado de tal forma que se realice la lectura en una sola línea:

❶

```
Inicio
    Leer (Nota_1, Nota_2, Nota_3)
    Promedio ←Nota_1+Nota_2+Nota_3) / 3
    Escribir (Promedio)
Fin
```

❶ **Leer** (*Nota_1, Nota_2, Nota_3*) Al utilizar esta instrucción se le está informando a la computadora que se van a ingresar tres valores por medio del teclado y que seguidamente estos valores ingresados debe de ser asignados a la variables *Nota_1, Nota_2 y Nota_3* respectivamente. (Respetando el orden)

Si es que ingresáramos por el teclado los números 4, 8 y 9, la computadora los asignará a las variables *Nota*_1, *Nota*_2 y *Nota*_3 respetando el orden en el que fueron ingresados dichos números, lo cual equivaldría a realizar las siguientes asignaciones:

*Nota*_1 ← 4
*Nota*_2 ← 8
*Nota*_3 ← 9

5.2.2. INSTRUCCIÓN DE ESCRITURA

La instrucción de escritura o también conocida como **salida de datos** permite mostrar valores que en algún momento se han ingresado a la computadora o que se hayan obtenido como consecuencia de haber realizado cálculos, operaciones, asignaciones, etc. (En otras palabras, permite la salida de información al exterior). De los dispositivos que permiten la salida de información de la computadora, los más usuales son el monitor, la impresora y los parlantes

siendo el más común el monitor. En el presente curso vamos a considerar exclusivamente al **monitor** como dispositivo de salida.

Sintaxis para la instrucción de Escritura:

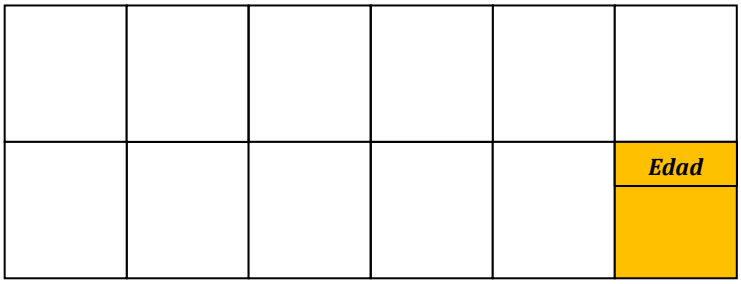
Escribir (*variable*)

Al utilizar la instrucción de escritura de una variable, debemos asegurarnos que esta contenga un valor.

Ejemplo: Se declara una variable de tipo entero.

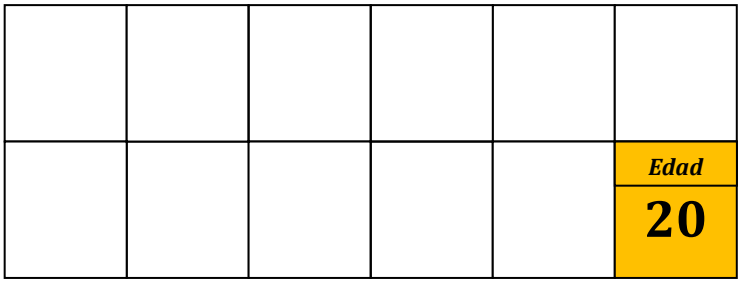
Var *Edad*: entero

Con la declaración se logra reservar un espacio de la memoria.



Luego se realiza la siguiente instrucción de asignación para almacenar directamente el valor entero 20.

Edad ← *20*



La instrucción que permite mostrar por la pantalla del monitor al valor almacenado en la variable *Edad* es la siguiente:

Escribir (*Edad*)

Para el caso en el que se tengan muchas variables podemos hacerlo de dos maneras:

* Escribiendo cada variable por separado:

```
Escribir (variable_1)
Escribir (variable_2)
        .
        .
        .
Escribir (variable_N)
```

* Utilizando una sola instrucción de escritura para todas las variables:

```
Escribir (variable_1, variable_2, ..... , variable_N)
```

Ejemplo: Continuemos analizando el ejemplo en el que desarrollamos el algoritmo para calcular el promedio de tres notas:

```
Inicio
    Leer (Nota_1)
    Leer (Nota_2)
    Leer (Nota_3)
    Promedio ←Nota_1+Nota_2+Nota_3) / 3
1   Escribir (Promedio)
Fin
```

En este algoritmo se puede observar que se está realizando una instrucción de escritura.

1 **Escribir** (*Promedio*) Al utilizar esta instrucción se le está informando a la computadora que debe mostrar por la pantalla del monitor el valor que se encuentra almacenado en la variable Promedio luego de haberse realizado la operación (*Nota_1*+*Nota_2*+*Nota_3*) / 3

De esta manera, si las variables: *Nota*_1, *Nota*_2, *Nota*_3 tienen almacenado los números 4, 8 y 9 respectivamente, la computadora los suma (4+8+9 = 21) y luego divide lo obtenido entre tres (21/3 = 7). El resultado final lo asigna a la variable *Promedio* y este es mostrado por la pantalla.

Debido a que en este curso solo escribiremos nuestros algoritmos usando lápiz y papel, tendremos que imaginarnos la manera cómo se muestran los resultados en pantalla.

5.2.3. PRESENTACIÓN DE DATOS EN PANTALLA

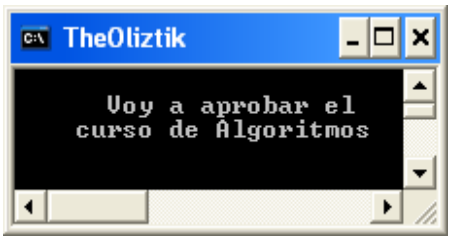
MENSAJES EN PANTALLA

- Con la instrucción de Escritura podemos mostrar mensajes en la pantalla del monitor con el objetivo de brindar mayor información al usuario que haga uso del programa. Para esto haremos uso de la instrucción de escritura mediante la palabra reservada **Escribir** seguida de paréntesis y comillas. El mensaje irá dentro de las comillas.

Escribir ("*Aquí va el mensaje*")

Ejemplo:

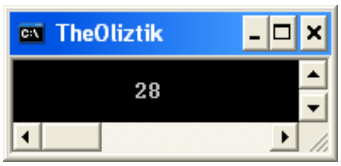
Escribir ("*Voy a aprobar el curso de Algoritmos*")



- Cuando deseamos mostrar el valor de una variable, podemos presentarla solitaria o acompañada de un mensaje que la describa.

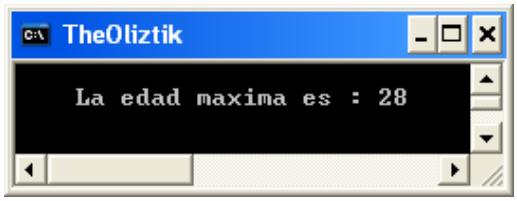
  Ejemplo: Supongamos que tenemos la variable entera *max_edad* que almacena el valor entero 28. Para presentar el valor que almacena esta variable usaríamos normalmente la siguiente instrucción:

Escribir (*max_edad*)



Podríamos brindar una información más completa si lo hacemos de la forma siguiente:

Escribir ("*La edad máxima es: "*, *max_edad*")



Se puede observar que lo que se muestra en la pantalla es el texto escrito entre comillas y el valor de la variable que se encuentra después de la coma.

Escribir ("La edad maxima es : ", max_edad)

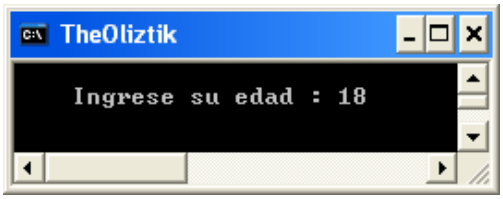
Se va a mostrar por la pantalla el texto escrito entre comillas.

Se mostrará por la pantalla el valor almacenado en la variable y no su identificador

- En el futuro cuando se encuentren programando (luego de haber aprobado el curso de algoritmos), tendrán la necesidad de recurrir al uso constante de los mensajes para que brinden mayor información y en algunos casos para que su programa resulte ser más **interactivo**; antes de una ***instrucción de lectura*** será necesario utilizar una ***instrucción de escritura*** en la que se envíe un mensaje indicando qué es lo que se va a introducir.

Ejemplo:

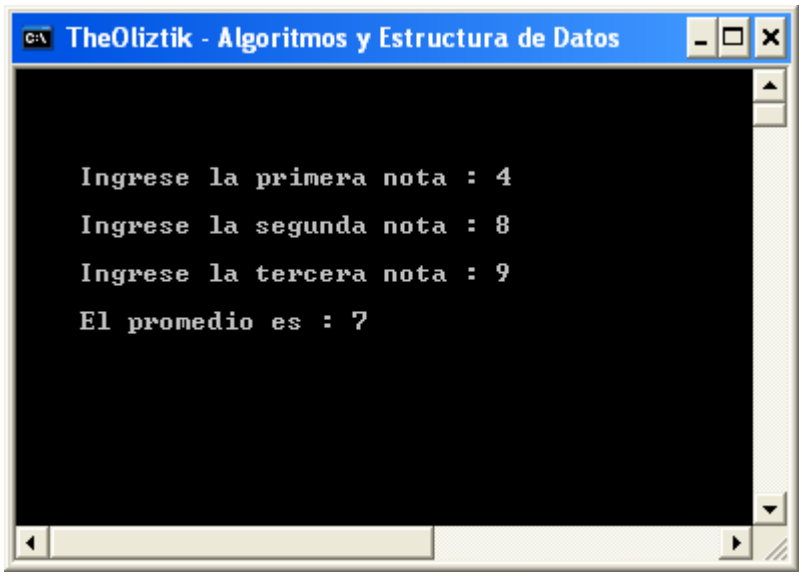
```
Escribir ("Ingrese su edad: ")
Leer (edad)
```



Ejemplo: Retornemos al problema en el que debíamos calcular el promedio de tres notas.

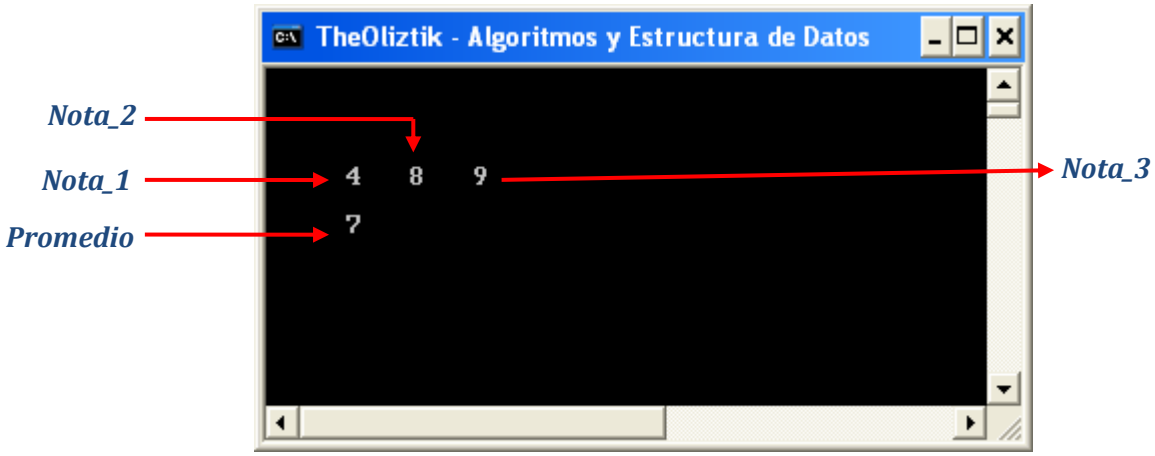
* El algoritmo de forma más interactiva sería:

```
Inicio
    Escribir ("Ingrese la primera nota : ")
    Leer (Nota_1)
    Escribir ("Ingrese la segunda nota : ")
    Leer (Nota_2)
    Escribir ("Ingrese la tercera nota : ")
    Leer (Nota_3)
    Promedio ← (Nota_1+Nota_2+Nota_3) / 3
    Escribir ("El promedio es : ", Promedio)
    Fin
```



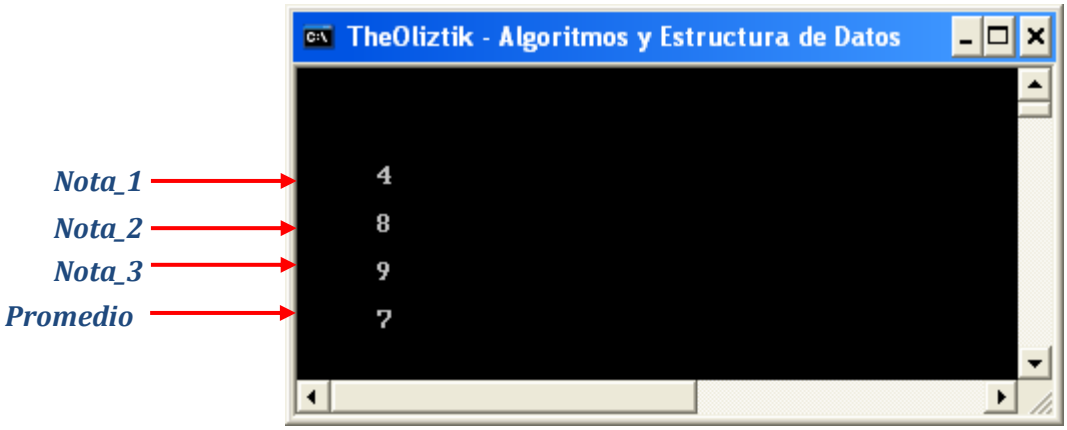
Durante este curso vamos a considerar opcional el uso de los mensajes y en la mayoría de los ejercicios realizaremos la lectura o escritura de datos sin acompañarlo de mensajes (sobreentendiendo que ya se sabe lo que se va a introducir) con el objetivo de ahorrar tiempo y de preocuparnos principalmente en cómo obtener la solución, ya que como veremos en los temas siguientes de este curso, obtener la solución de un problema no es tan fácil y requiere de análisis y tiempo. Así que nuestro algoritmo simplemente será de la siguiente manera:

Inicio
    Leer (*Nota_1, Nota_2, Nota_3*)
    Promedio ← (*Nota_1*+*Nota_2*+*Nota_3*) / 3
    Escribir (*Promedio*)
Fin



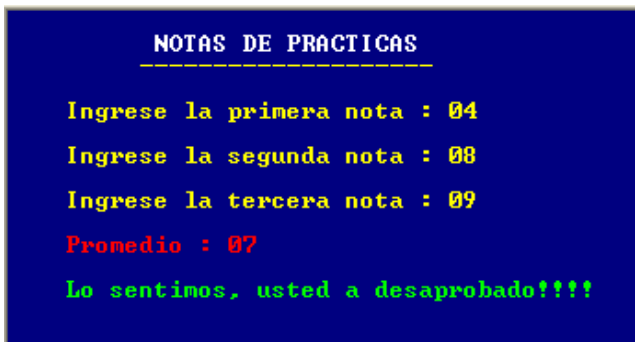
O así también:

```
Inicio
    Leer (Nota_1)
    Leer (Nota_2)
    Leer (Nota_3)
    Promedio ← (Nota_1+Nota_2+Nota_3) / 3
    Escribir (Promedio)
Fin
```



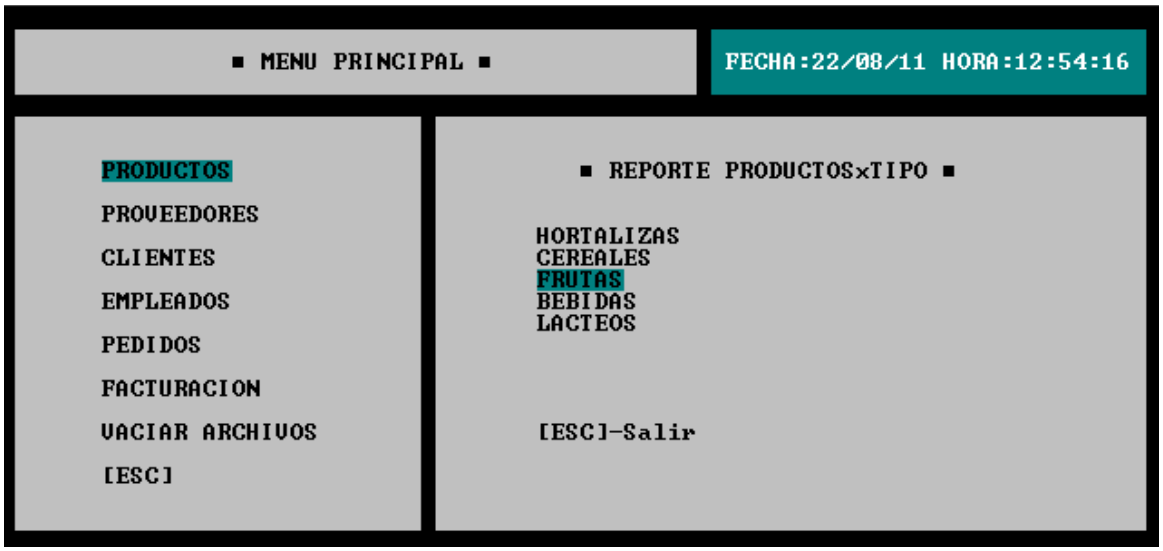
Una vez que hayan aprobado el curso de algoritmos y se encuentren programando podrán mejorar sus programas añadiendo otras cosas triviales como color y ciertos efectos lo cual en algoritmos no tiene ningún sentido tratar.

Ejemplo: Programa que calcula el promedio de tres notas de prácticas.



Ejemplo: Programa que simula un Sistema de Matrícula en la UNI FIIS.

Ejemplo: Programa para un negocio.



6. VARIABLES II

Las variables van a permitir el almacenamiento de valores de forma temporal en la memoria mientras el programa se encuentre en ejecución y van a ser utilizados para diferentes fines. Según su finalidad, las podemos agrupar en 5 categorías principales:

- ✓ Variable de trabajo
- ✓ Acumulador
- ✓ Contador
- ✓ Variable de bandera
- ✓ Variable auxiliar

6.1. VARIABLE DE TRABAJO:

Llamaremos así a las variables que usaremos más a menudo ya que en ellas almacenaremos todos los valores de los datos de entrada así como también los valores de los datos de salida y aquellos valores que resulten de alguna operación.

En el ejemplo en donde se desarrolló el algoritmo para calcular el promedio de tres notas se usaron las siguientes variables de trabajo: ***Nota_1***, ***Nota_2***, ***Nota_3*** y ***Promedio.***

Las tres primeras variables permitieron almacenar los valores de entrada.

La cuarta variable recibió el resultado que se obtuvo luego de haberse operado las tres variables iniciales lo cual vendría ha ser el valor del dato de salida.

6.2. ACUMULADOR:

También llamada **Sumador.** Son variables que usaremos para almacenar la suma acumulativa de un conjunto de valores que se van leyendo o calculando continuamente. Los acumuladores generalmente se incrementan o disminuyen en una cantidad variable. Se usarán especialmente cuando realicemos operaciones repetitivas.

Usaremos la siguiente asignación:

acumulador ← *acumulador* + expresión_numérica

Ejemplo:

Consideremos ***S*** y ***num*** variables enteras, donde el acumulador sería *S*

Asignémosle un primer valor a la variable **S**

S ← 0

Asignemos a continuación un valor a la variable entera ***num*.**

$num \leftarrow 20$

$S \leftarrow S + num$

(El acumulador S tendrá almacenado ahora el resultado de sumar 0 + 20 , es decir, 20)

$num \leftarrow 18$

(la variable *num* ahora tendrá como valor al número 18)

$S \leftarrow S + num$

(el acumulador S almacenará ahora el resultado de sumar 20 + 18, es decir, 38)

Inicialización de un acumulador: Se llama así al hecho de asignar por primera vez un valor inicial a un acumulador**.** Generalmente los acumuladores se inicializan con el valor numérico cero.

NOTA:

- ✓ Los acumuladores son variables que almacenan cantidades numéricas, por lo tanto solo pueden ser de tipo **real** o de tipo **entero**.

6.3. CONTADOR:

Son variables enteras que permiten llevar el control de un número de eventos o sucesos. Su valor se incrementa o decrementa en una cantidad fija, generalmente de uno en uno. En forma similar al acumulador, los contadores se usarán principalmente cuando realicemos operaciones que se repiten continuamente (bucles).

Usaremos la siguiente asignación:

$contador \leftarrow contador + 1$

Un nombre muy usual que se le da a esta variable es ***cont***, pero puede ser identificado con cualquier otro nombre.

Ejemplo: Consideremos la variable *cont* entera

```
cont ← 0
```

```
cont ← cont +
```

(Ahora el contador tendrá como valor el resultado de sumar 0 + 1, es decir, 1)

```
cont ← cont +
```

(Ahora el contador tendrá como valor el resultado de sumar 1 + 1, es decir, 2)

Inicialización de un contador: Se llama así al hecho de asignar por primera vez un valor inicial a un contador**.** Los contadores, dependiendo de la forma del algoritmo serán inicializados con el valor numérico uno o cero.

NOTA:

- ✓ Los acumuladores son variables que almacenan cantidades numéricas enteras, por lo tanto solo pueden ser de tipo **entero**.

6.4 VARIABLE DE BANDERA:

Son variables que almacenan valores de tipo lógico por lo tanto pueden asumir uno de dos valores: Verdadero y Falso. Serán usadas en estructuras en las que se requiera analizar condiciones. Las variables de bandera reciben otros nombres: **variable interruptor, variable indicador, variable centinela, variable conmutador, variable switch, etc.**

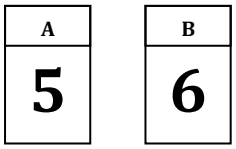
6.5 VARIABLE AUXILIAR:

Son variables que nos serán de ayuda en muchos casos para almacenar los valores de otras variables ya existentes con la finalidad de no perderlos.

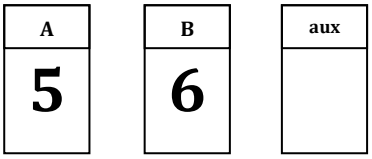
Ejemplo: Cuando queremos intercambiar los valores almacenados en dos variables nos veremos en la necesidad de utilizar una variable auxiliar.

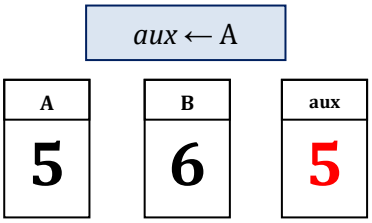
En un inicio la variable A almacena el valor 5 y la variable B almacena el valor de 6.

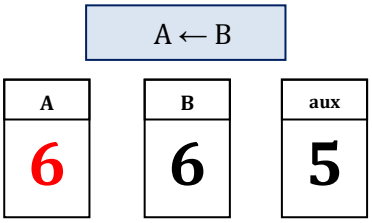
```
A ← 5
B ← 6
```

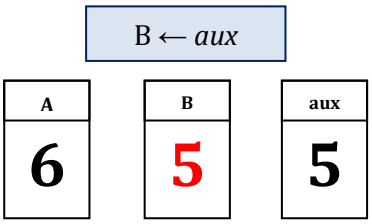


Para intercambiar sus valores debemos de hacer uso de una variable auxiliar y realizar tres asignaciones de la forma siguiente:









7. ESTRUCTURA DEL PSEUDOCÓDIGO

En esta parte del curso se presenta la estructura que debe tener el pseudocódigo para representar los algoritmos. Esta estructura va a permitir organizar todos los elementos que aparecen en este. Esta estructura es similar a la de los Lenguajes de Programación.

Partes:

- ✓ Encabezado o Cabecera
- ✓ Secciones
  - Sección de creación de Tipos
  - Sección de declaración de Constantes y Variables
  - Sección para los subprogramas
- ✓ Cuerpo de Acciones Ejecutables

7.1. ENCABEZADO O CABECERA

Aquí vamos a escribir la palabra reservada **Pseudocódigo** y seguidamente escribiremos un identificador válido para el algoritmo, es decir el nombre del algoritmo. (O nombre del programa), el cual, se sugiere que sea un nombre que lo describa.

Sintaxis:

Pseudocódigo Identificador_Algoritmo ← **Encabezado o Cabecera**

Ejemplo:

Pseudocódigo Promedio

Pseudocódigo Prom_2

Pseudocódigo The_Oliztik

Otros autores en lugar de utilizar la palabra **Pseudocódigo** en el encabezado prefieren usar la palabra **Algoritmo**, como es el caso de la profesora Grimanesa. Ejemplo:

Algoritmo Identificador_Algoritmo

7.2 SECCIONES

- ✓ Sección de creación de Tipos.
- ✓ Sección de declaración de constantes y variables.
- ✓ Sección para los subprogramas.
- ✓ Acciones ejecutables.

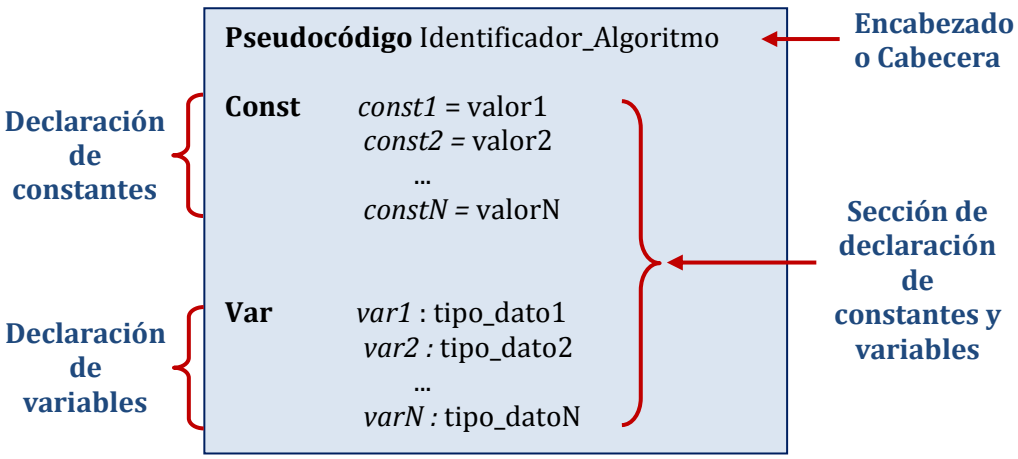
7.2.1. Sección de creación de Tipos

Lo relacionado a los **Tipos** se verá en temas posteriores por lo que por ahora esta sección permanecerá vacía.

7.2.2. Sección de declaración de Constantes y Variables

Como se dijo en algún momento, para poder manipular variables es necesario **declararlas** (con el objetivo de reservar un espacio en la memoria) y esto se realiza en esta sección. Declarar una variable significa especificar el **nombre** o **Identificador** de la variable y el **Tipo** de valores que puede almacenar. Esto es muy importante y debe realizarse antes de comenzar con el cuerpo de acciones ejecutables. Con las constantes no ocurre lo mismo pues no necesitan ser declaradas para su posterior uso; pero si es que optamos por declararlas, lo haríamos dándoles un nombre y un valor. La declaración se realizará debajo del encabezado y para ello usaremos las palabras reservadas **Const** y **Var** (en ese orden).

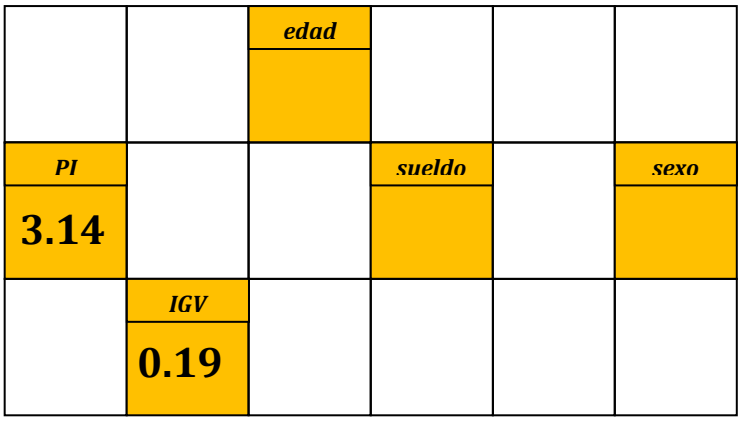
Sintaxis:



Ejemplo: Si se desea utilizar a las constantes PI (número irracional) e IGV (impuesto general a las ventas) y a las variables *edad*, *sueldo, sexo*, las declararemos de la siguiente manera:

Const	PI = 3.14 IGV = 0.19
Var	*edad:* entero *sueldo:* real *sexo*: carácter

En la memoria se reservarán 5 espacios de memoria (casilleros) para las 2 constantes y tres variables declaradas.



La variable *edad* se declara de tipo entera pues almacenará edades los cuales son cantidades enteras. La variable *sueldo* se declara de tipo real porque almacenará valores numéricos con punto decimal (reales) y la variable *sexo* se declara de tipo carácter para que almacene el carácter ' M ' cuando se trate del género masculino o ' F ' para el género femenino.

OBSERVACIONES:

- ✓ La declaración de variables es imprescindible. Al momento de la declaración de una variable se especifica su **Identificador** y su **Tipo**. Los valores que almacenará le serán asignados luego, en los pasos del algoritmo.
- ✓ La declaración de constantes no siempre es necesaria. Como podrán apreciar en los ejemplos posteriores de este curso en la mayoría no se declararán constantes. Si es que se opta por declarar una constante, se debe especificar su **Identificador** y allí mismo se le asigna un **Valor** el cual no cambiará en ningún paso del algoritmo. La computadora reconocerá a que **Tipo** pertenece según el valor que se le asigne.

Además, el operador para inicializar una constante no es el operador **flecha** ← sino será el operador **igualdad =**

Ejemplo: Declare tres variables de tipo entero, dos variables de tipo real y 1 de tipo carácter. Además declare una constante cuyo valor sea 2.7281

El identificador de las variables queda a criterio de cada persona. En este caso las variables enteras tendrán como identificador: n1, n2, n3; las variables reales serán llamadas: R1 y R2; y la variable carácter será llamada: opc. La constante la identificaremos como **e**.

- La declaración sería la siguiente:

```
Const      e = 2.7281

Var        n1 : entero
           n2 : entero
           n3 : entero
           R1 : real
           R2 : real
           opc : caracter
```

- Cuando existan variables del mismo tipo, estas se pueden declarar en grupos (lo cual es lo más conveniente) como veremos a continuación:

```
Const      e = 2.7281

Var        n1, n2, n3 : entero
           R1, R2 : real
           opc : caracter
```

- No existe un orden único en cuanto a la declaración de variables. Lo siguiente también es correcto:

```
Const      e = 2.7281

Var        R1, R2 : real
           n1, n2 : entero
           opc : carácter
           n3 : entero
```

OTROS FORMATOS DE DECLARACIÓN

Al ser el pseudocódigo un lenguaje informal, no existe un patrón único para su escritura, por lo que diversos autores utilizan el formato que más les parece cómodo.

Ejemplo:

Def. Variables:
edad: **entero**
sueldo: **real**
sexo: **caracter**
CONST
PI = 3.1416

Ejemplo:

Variables
entero: *edad*
real: *sueldo*
caracter: *sexo*

Durante todo este libro usaremos el primer formato presentado.

7.2.3. Sección para los subprogramas

Por el momento no será utilizado.

7.3. ACCIONES EJECUTABLES

En este bloque se va a realizar todo el proceso de resolución de algún problema mediante un conjunto de pasos los cuales reciben el nombre de **acciones** o **instrucciones**. (Estrictamente se llama **acción** cuando se trata de un algoritmo y se le conoce como **instrucción** cuando se trata de un programa, en este texto son considerados sinónimos)

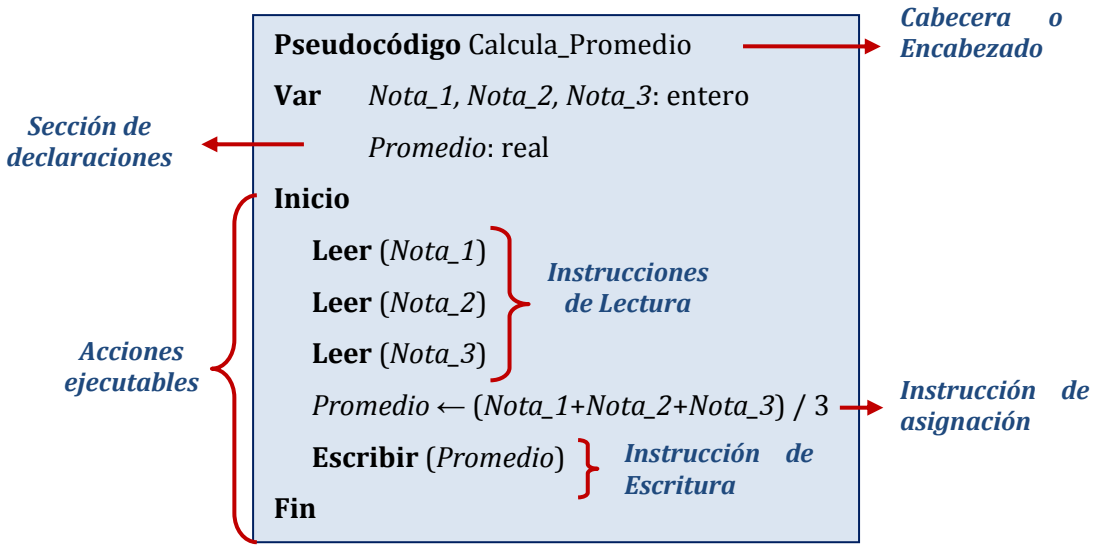
Este bloque debe tener escrito en su primera línea la Palabra Reserva **Inicio.** Al final del bloque debemos escribir la Palabra Reservada **Fin.** Estas palabras indican el comienzo y el término del algoritmo. Para una mayor legibilidad se utiliza el sangrado.

```
Inicio
    Instrucción 1
    Instrucción 2
    Instrucción 3
          ....
    Instrucción n
Fin
```

En resumen, la estructura general de todo pseudocódigo será el siguiente:

```
Pseudocódigo Identificador_Algoritmo

Tipo

Const     const1 = valor1
          const2 = valor2
             ...
          constN = valorN

Var       var1 : tipo_dato1
          var2 : tipo_dato2
             ...
          varN : tipo_datoN

  SUBPROGRAMAS

Inicio
    Instrucción 1
    Instrucción 2
    Instrucción 3
          ....
    Instrucción n
Fin
```

Ejemplo: Recordemos nuevamente nuestro algoritmo que permitía hallar el promedio de 3 notas y presentemos ahora la estructura del pseudocódigo en forma completa:



DOCUMENTACIÓN Y COMENTARIOS

- En ocasiones podemos hacer uso de comentarios internos. Estos permiten explicar a las personas que vayan a revisar nuestro pseudocódigo el porqué se realizan ciertas acciones y además permiten que sean fáciles de interpretar, de comprender y de modificar.
- Para la realización de comentarios, se presentan las siguientes alternativas:

Sintaxis para los comentarios

```
// Aquí va su comentario
```

```
/* Aquí va su comentario */
```

```
(* Aquí va su comentario *)
```

- El uso de los comentarios no es imprescindible pero sí muy recomendable por las razones ya expuestas.
- Los comentarios no van a aparecer en la pantalla ni tampoco van a interferir con las instrucciones, solo son **ayudas visuales**.

UNIDAD II

INTRODUCCIÓN: *Un Poco De Historia*

Remontémonos a inicios de la década de los 60. Fue una época en donde no existía una metodología al momento de programar. Cada quién lo hacía como podía sin orden, ni criterio, ni planificación previa.

En un primer momento, la ausencia de una metodología de programación no era un problema por la poca difusión, la baja potencia de las computadoras y las necesidades poco ambiciosas de la época, pero esto fue cambiando ya que las máquinas se iban generalizando, éstas se hacían cada vez más potentes, y más programadores se iban incorporando a la profesión, por ende, las aplicaciones aumentaban tanto en número como en complejidad. De esta manera los programas se iban volviendo más complicados lo cual traía serios problemas al programarlos y más aún a la hora de depurarlos (corregirlos), mantenerlos, modificarlos y actualizarlos. Por todo esto surge la necesidad de una manera ordenada y sistemática a la hora de programar. Como resultado de esta necesidad nace la P**rogramación Estructurada.**

Surgimiento de la Programación Estructurada

La programación estructurada es un conjunto de técnicas usadas en la programación que han ido evolucionando de tal manera que aumentan la productividad del programador reduciendo el tiempo requerido para escribir, verificar, depurar y mantener los programas. La programación estructurada está basada en el **Teorema de la Programación Estructurada** el cual proporciona las bases sobre las que ésta se sustenta. Este teorema nos dice que cualquier programa **propio**, sin importar su complejidad, puede ser construido utilizando combinaciones de tres estructuras de control de flujo: Secuenciales, Selectivas y Repetitivas. El Teorema de la programación estructurada fue demostrado por Botin y Jacopeni en un artículo publicado en mayo de 1966.

Así se llegó a demostrar que con tres estructuras de control se pueden escribir todos los programas y aplicaciones posibles.

8. ESTRUCTURAS DE CONTROL DE FLUJO

FLUJO DE UN PROGRAMA

El flujo de un programa es el orden de ejecución de sus instrucciones. (El flujo de un algoritmo es el orden de realización de sus acciones)

Las estructuras de control de flujo tienen la cualidad de llevar un orden, en algunos casos de tomar decisiones y en muchas ocasiones de repetir un conjunto de instrucciones un número finito de veces. El aprendizaje de las estructuras de control es sumamente importante, ya que es a partir de estas tres estructuras que vamos a solucionar todos los problemas. Son estas tres estructuras la columna vertebral de la programación estructurada.

I. Estructuras Secuenciales
II. Estructuras Selectivas
III. Estructuras Repetitivas

8.1. ESTRUCTURAS SECUENCIALES

Las estructuras secuenciales son las estructuras de control más simples. Son aquellas en las que las instrucciones se ejecutan una a continuación de otra siguiendo siempre una única secuencia. La salida de una acción es la entrada de la siguiente y así sucesivamente, hasta el final del proceso.

Sintaxis:

```
Pseudocódigo Nomb_Alg
Const
Var
Inicio
    Acción 1
    Acción 2
    Acción 3
    ...
Fin
```

Ejemplo: *Diseñe un algoritmo que permita mostrar por la pantalla del monitor el mensaje siguiente: "Hola Mundo".* **Solución:**

Análisis del Problema:
Datos de Entrada: Ninguno

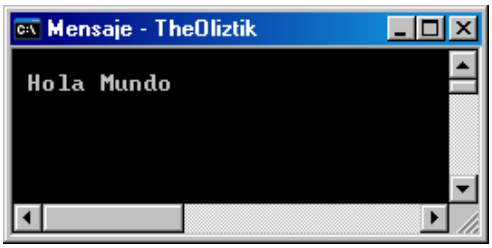
Datos de Salida: El mensaje: "*Hola mundo*"

Proceso: Instrucción de escritura.

Diseño de la Solución:

Pseudocódigo Mensaje

Inicio
    Escribir ("*Hola Mundo* ")
Fin



Ejemplo: *Diseñe un algoritmo que permita calcular la suma de dos números enteros los cuales deben de ser ingresados por el teclado*. **Solución:**

Análisis del Problema:

¿Datos de entrada?	Los números que se van a sumar: a, b
¿Datos de salida?	La suma de los números: c
Proceso:	c ← a + b

Diseño de la solución:

Pseudocódigo Suma_números

Var *a, b, c*: entero
Inicio
    Leer (*a, b*)
    c ← *a* + *b*
    Escribir (*c*)
Fin

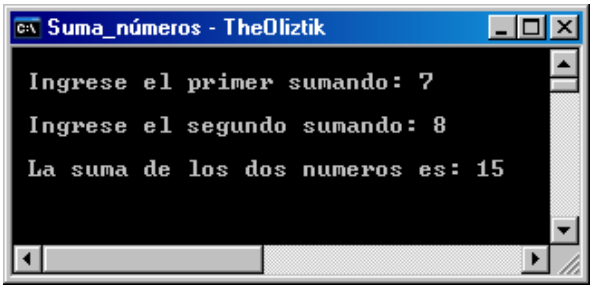
Se dijo anteriormente que antes de la instrucción de lectura lo correcto sería que vaya una instrucción de escritura que envíe un mensaje por la pantalla del monitor en el cual pregunte qué datos se van a ingresar, para luego recién leerlos. Pero también dijimos que esto se va a obviar en la mayoría de los ejercicios de algoritmos para poder concentrarnos en lo que es más importante por ahora, el conjunto de acciones que permiten obtener la solución.

El pseudocódigo de forma más interactiva (con mensajes en pantalla que permitan ofrecer mayor información) sería de la siguiente manera:

Pseudocódigo Suma_números

Var *a, b, c*: entero

Inicio
Escribir ("*Ingrese el primer sumando*: ")
Leer (*a*)
Escribir ("*Ingrese el segundo sumando*: ")
Leer (*b*)
$c \leftarrow a + b$
Escribir ("*La suma de los dos números es*: ", *c*)
Fin



La interactividad no nos debe preocupar por ahora ya que esto será tratado más ampliamente en **Programación**. Por ahora la prioridad es aprender a realizar algoritmos de forma eficiente.

Ejemplo: *Diseñe un algoritmo que permita calcular el producto de dos números enteros los cuales deben de ser ingresados por el teclado.* **Solución:**

Análisis del Problema:

¿Datos de entrada?	Los números que se van a multiplicar: a, b
¿Datos de salida?	El producto de los números: c
Proceso:	c ← a * b

Diseño de la solución:

Pseudocódigo Producto_números

Var *a, b, c*: entero

Inicio
  Leer (*a*, *b*)
  c ← *a* * *b*
  Escribir (*c*)
Fin

Podemos disminuir el uso de variables modificando nuestro algoritmo de la manera siguiente:

Pseudocódigo Producto_números

Var *a, b*: entero

Inicio
  Leer (*a*, *b*)
  Escribir (*a*b*)
Fin

Ejemplo: *Si se define el operador ☺ de la siguiente manera:*

$$a \☺ b = (a+b)\ (a-b)$$

Diseñe un algoritmo que permita calcular el resultado de operar dos números enteros (ingresados por el teclado) mediante el operador definido anteriormente. **Solución:**

Análisis del Problema:

¿Datos de entrada?	Los números que se van a operar: a , b
¿Datos de salida?	El resultado de haber operado: c
Proceso:	c ← (a + b) * (a - b)

Diseño de la solución:

Pseudocódigo Operación_números

Var *a, b, c*: entero

Inicio
  Leer (*a*, *b*)
  c ← (*a* + *b*) * (*a* - *b*)
  Escribir (*c*)
Fin

Si deseamos prescindir de la variable *c* haríamos lo siguiente:

Pseudocódigo Operación_números

Var *a, b:* entero

Inicio
  Leer (*a*, *b*)
  Escribir ((*a* + *b*) * (*a* - *b*))
Fin

De esta manera disminuimos el número instrucciones y el número de variables.

Ejemplo: ***Diseñar un algoritmo que obtenga el valor de "y" a partir de la ecuación $y=3x^2+7x-15$ solicitando como dato de entrada el valor de x.*** **Solución:**

Análisis del Problema:

¿Datos de entrada? *x*

¿Datos de salida? *y*

Proceso: *y ← 3 * x * x + 7 * x – 15*

Diseño de la solución:

Pseudocódigo Ecuación

Var *x, y*: real

Inicio

Leer (*x*)

*y ← 3 * x * x + 7 * x – 15*

Escribir (y)

Fin

Antes de la instrucción de lectura:

Leer (x)

Pudimos haber enviado un mensaje de la siguiente manera:

Escribir ("Ingrese el valor de x").

En el curso de algoritmos, el uso de estos mensajes será opcional, ha no ser que el profesor indique lo contrario.

Ejemplo: ***Diseñe un algoritmo para calcular el salario neto de un trabajador si se conoce el número de horas trabajadas, precio de la hora de trabajo y considerando unos descuentos fijos, el sueldo bruto en concepto de impuestos (20%).*** **Solución:**

Vamos a declarar la constante **Impuesto** con el valor 0.20

Pseudocódigo Salario

Const *Impuesto* = 0.20

Var *horas_trab, precio_hora, sue_bru, sue_ne*: real

Inicio

Leer (*horas_trab, precio_hora*)

*sue_bru ← horas_trab * precio_hora*

sue_net ← **Impuesto** * *sue_bru*

Escribir (*sue_ne*)

Fin

Como se dijo anteriormente, en los ejercicios no será necesaria la declaración de constantes ya que podemos usarlas sin la necesidad de darles un nombre. Así el problema se puede resolver más cómodamente de la siguiente manera:

Pseudocódigo Salario

Var *horas_trab, precio_hora, sue_bru, sue_ne*: real

Inicio

Leer (*horas_trab, precio_hora*)

sue_bru ← *horas_trab* * *precio_hora*

sue_net ← 0.8 * *sue_bru*

Escribir (*sue_ne*)

Fin

Aquí estamos haciendo uso de la constante real 0.8 de forma directa sin la necesidad de haberla declarado (dándole un nombre)

Ejemplo: ***Diseñar un algoritmo para calcular el monto que debe pagar un cliente que desea comprar los productos A, B y C. Los precios de venta se dan como dato. Existe un descuento del 10% para los productos A y B antes del I.G.V.*** **Solución:**

Pseudocódigo Ejemplo

Var *PA, PB, PC, CA, CB, CC, PTA, PTB, PTC, IGV, PP*: real

Inicio

Escribir ("*Ingrese los precios de los productos A, B y C*")

Leer (*PA, PB, PC*)

PTA ← 0.9 * (*PA * CA*)

PTB ← 0.9 * (*PB * CB*)

PTC ← *PC * CC*

IGV ← 0.19 * (*PTA+PTB+PTC*)

PP ← *PTA+PTB+PTC+IGV*

Escribir ("*Total a pagar: "*, *PP*)

Fin

Este mensaje será opcional

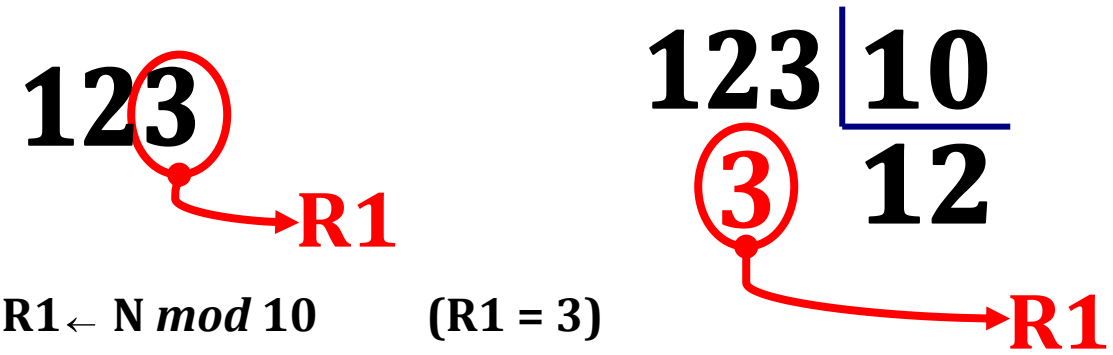
APLICACIONES DE LOS OPERADORES DIV y MOD

Los operadores *div* y *mod* resultan ser útiles en muchos problemas relacionados a la inversión de números, cálculo del máximo común divisor, verificación de numerales capicúas, cambios de sistema de numeración, cantidad de divisores de un número, verificar si un número es primo, . . , etc.

Ejemplo: Pasos para la inversión de un número cuando se conoce su cantidad de cifras.

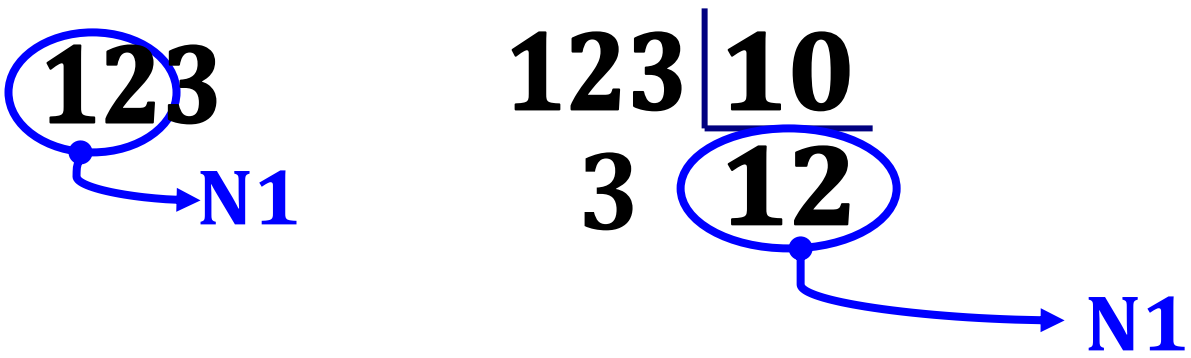
Sea **N = 123**

1. Debemos *tomar prestado* la cifra que se encuentra a la derecha de 123, el cual es 3, para eso dividimos 123 entre 10 nos quedamos con el residuo de la división entera el cual lo almacenamos en una variable llamada R1. (Utilizamos el operador *mod*)



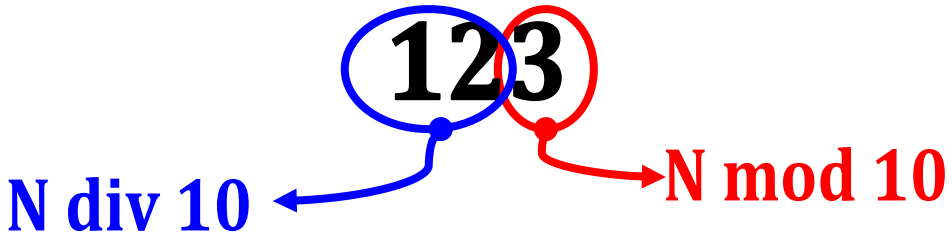
R1 ← N *mod* 10 **(R1 = 3)**

2. Ahora debemos borrar al número 3 que pertenece a 123 para quedarnos con el número que queda a la izquierda del 3, es decir, con 12. Para eso dividimos 123 entre 10 y nos quedamos con el cociente el cual lo almacenamos en una variable llamada N1. (Utilizamos el operador *div*)

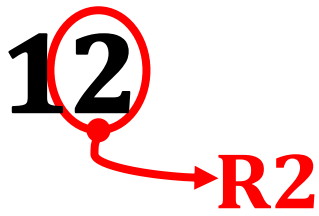


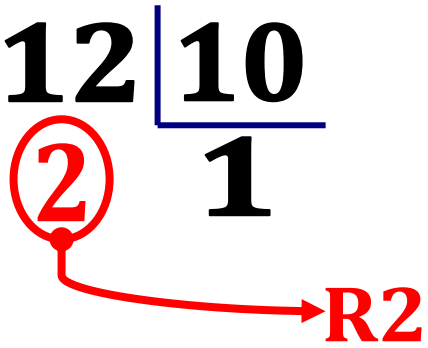
N1 ← N *div* 10 **(N1=12)**

Hasta esta parte hemos obtenido lo siguiente:



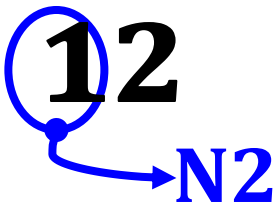
3. Volvemos a repetir los pasos anteriores, tomaremos la primera cifra de la derecha de 12 el cual es 2, por eso dividimos 12 entre 10 y nos quedamos con el residuo el cual lo almacenamos en una variable llamada R2. (Utilizamos el operador *mod*)

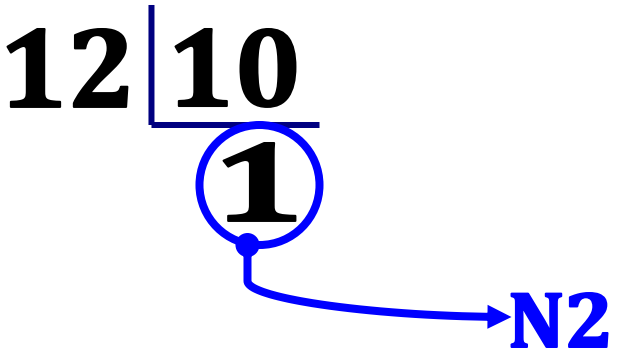




R2 ← N1 *mod* 10 (R2 = 2)

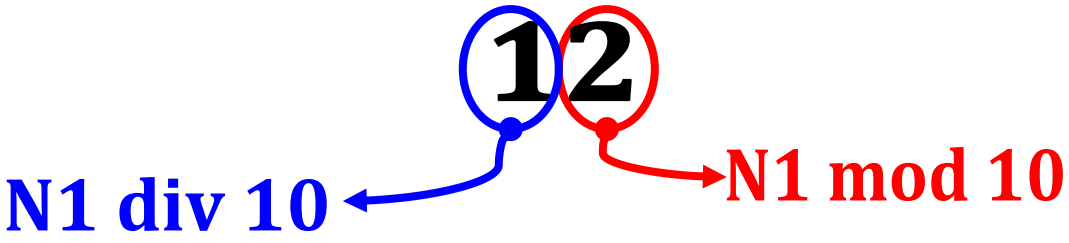
4. Ahora debemos quedarnos con el número que queda a la izquierda de 2, es decir, con 1. Para eso dividimos 12 entre 10 y nos quedamos con el cociente el cual lo almacenamos en una variable llamada N2. (Utilizamos el operador *div*)



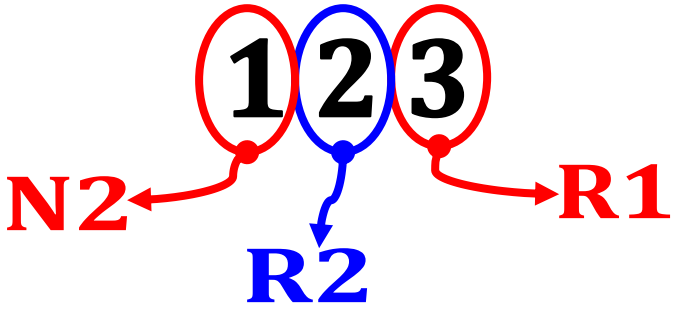


N2 ← N1 *div* 10 (N2=1)

Así tenemos:



Ahora que ya tenemos los dígitos del número inicial N = 123 almacenados en distintas variables, ya podremos formar el número invertido.



Sea la variable NI la que almacene al número invertido. Se tiene:

NI ← R1 * 100 + R2 * 10 + N2

Ejemplo: ***Diseñar un algoritmo para invertir un número entero de 4 cifras.***

Nota: *Pondremos en práctica la utilización de los operadores* ***div*** *y* ***mod.*** **Solución:**

¿Datos de entrada? Número de 4 dígitos: *N (Debe de ser positivo)*

¿Datos de salida? Número invertido: *NI*

¿Proceso? Divisiones sucesivas entre 10

Pseudocódigo Inversión

Var *N, N1, NI, R1, R2, R3, R4*: entero

Inicio

Leer (N)
R1 ← *N mod* 10
N1 ← *N div* 10
R2 ← *N1 mod* 10
N2 ← *N1 div* 10
R3 ← *N2 mod* 10
N3 ← *N2 div* 10
NI ← *R1* * 1000 + *R2* * 100 + *R3* * 10 + *N3*
Escribir ("*El número invertido es: "*,NI)

Fin

Ejemplo: ***Diseñar un algoritmo que permita recibir un número positivo cualquiera y que devuelva la cifra de las UNIDADES.***

Si el número es: 123 , el algoritmo debe devolver: 3

Solución:

¿Datos de entrada? Número *N*

¿Datos de salida? Cifra de las UNIDADES

¿Proceso? Resto de la división entera entre 10

Pseudocódigo Inversión

Var *Num, r*: entero

Inicio

Leer (*Num*)
r ← *Num* ***mod*** 10
Escribir (*r*)

Fin

8.2. ESTRUCTURAS SELECTIVAS

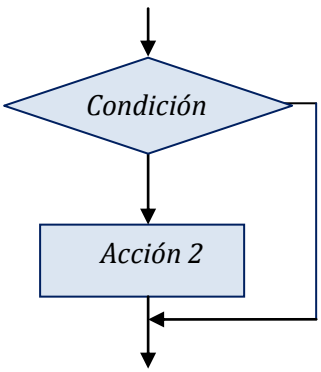
Son también llamadas ESTRUCTURAS CONDICIONALES, ESTRUCTURAS DE DECISIÓN o ESTRUCTURAS ALTERNATIVAS. Son usadas para tomar decisiones. En estas estructuras se evalúa una condición (*una condición es una expresión lógica que da como resultado o verdadero o falso*) y según el resultado que se obtiene se realiza una u otra acción.

CLASIFICACIÓN:

8.2.1. ESTRUCTURA SELECTIVA SIMPLE: (*Si ... entonces*) , también conocida como alternativa simple.

Se evalúa una condición (Expresión Relacional o Expresión Lógica) y:

- Si es verdadera ⇨ Se ejecutan ciertas acciones
- Si es falsa ⇨ No se ejecuta ninguna acción, continúa



Si (*condición*) **Entonces**
Acción 1
Acción 2
Acción 3
...
fin_si

Ejemplo: ***Diseñar un algoritmo que halle el nuevo sueldo de un empleado si conocido su sueldo actual se le aplique un aumento del 15% en el caso que dicho sueldo actual sea superior a S/. 1000.*** **Solución:**

Pseudocódigo Sueldo
Var *sueldo*: entero
Inicio
Escribir (*"Ingrese su sueldo actual"*)
Leer (*sueldo*)
Si (*sueldo* > 1000) **entonces**
sueldo ← **1.15** * *sueldo*
fin_si
Escribir (*sueldo*)
Fin

Ejemplo: ***Diseñar un algoritmo para calcular el promedio de 4 notas de prácticas calificadas eliminando la menor nota.***

Solución:

¿Datos de entrada? *Las 4 notas de las prácticas*
¿Datos de salida? *El promedio de las 3 notas más altas.*
¿Proceso? *Encontrar la menor nota mediante comparaciones.*

Pseudocódigo Promedio_práctica

Var *pc1, pc2, pc3, pc4, mín*: entero
prom: real

Inicio

Escribir (*"Ingrese las notas de prácticas"*)
Leer (*pc1, pc2, pc3, pc4*)
mín ← *pc1*
Si (*pc2* < *mín*) **entonces**
mín ← *pc2*
fin_si
Si (*pc3* < *mín*) **entonces**
mín ← *pc3*
fin_si
Si (*pc4* < *mín*) **entonces**
mín ← *pc4*
fin_si
prom ← (*pc1+pc2+pc3+pc4-mín*) ***/ 3***
Escribir ("*El promedio de prácticas es: "*, *prom*)

Inicializamos la variable *mín* con el valor de cualquiera de las prácticas: *pc1, pc2, pc3, pc4*, en este caso con *pc1*

Fin

Ejemplo: ***Se desea leer 3 números enteros y luego:***

a) Identifique y presente el número medio del conjunto de los 3 números.

b) Organice los 3 números en forma ascendente.

Solución:

Pseudocódigo Ejemplo

Var *n1, n2, n3, máx, mín, me*: entero

Inicio

Leer (*n1, n2, n3*)
máx ← *n1*
mín ← *n1*

Inicializamos las variables *max* y *mín* con el valor de cualquiera de las notas: *n1, n2, n3;* en este caso con *n1*

Si (*n2* > *máx*) **entonces**
máx ← *n2*
fin_si

Si (*n3* > *máx*) **entonces**
máx ← *n3*
fin_si

Si (*n2* < *mín*) **entonces**
mín ← *n2*
fin_si

Si (*n3* < *mín*) **entonces**
mín ← *n3*
fin_si

(Calculando el número medio del conjunto**)**

me ← *n1 + n2 + n3 – (máx + mín)*

Escribir (*me*)

(Mostrando los números en forma ascendente**)**

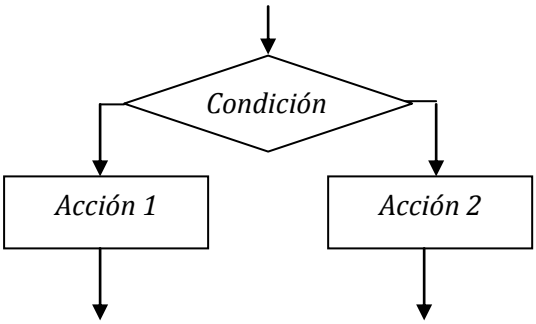
Escribir (*mín, me, máx*)

Fin

8.2.2. ESTRUCTURA SELECTIVA DOBLE: (***Si … entonces; Sino***) , también conocida como alternativa doble.

Se utiliza cuando se evalúa una condición (Expresión Relacional o Expresión Lógica) y se presentan dos alternativas, luego:

- Si la condición es verdadera ⇨ Se ejecutan ciertas acciones
- Si la condición es falsa ⇨ Se ejecutan otras acciones




```
Si (condición) Entonces
        Acciones1
Sino
        Acciones2
fin_si
```

Ejemplo: ***Una universidad aplica dos exámenes E1 y E2 a sus postulantes. Los postulantes son admitidos si tienen una calificación mayor que 80 en al menos uno de los exámenes, en caso contrario, será rechazado. Diseñe un algoritmo que solucione el problema y diga si un alumno está aprobado o desaprobado.***

Solución:

```
Pseudocódigo Examen_Admisión
Var     E1, E2: entero
Inicio
Leer (E1, E2)

   Si (E1 > 80  ∨  E2 > 80) entonces
        Escribir ("admitido")
   sino
        Escribir ("rechazado")
   fin_si

Fin
```

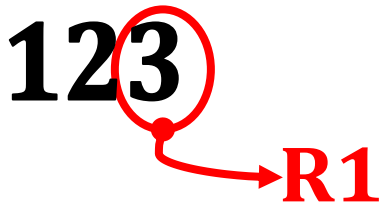
APLICACIÓN DE LOS OPERADORES DIV y MOD

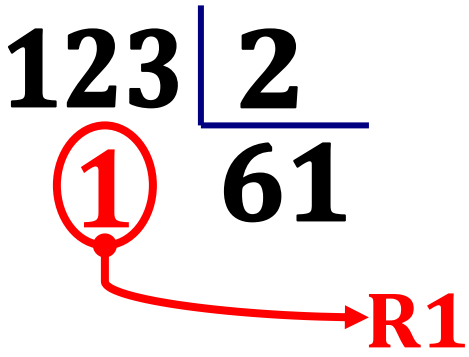
Ejemplo: Verificar la paridad de un número.

Un número es par cuando al ser dividido entre 2 su residuo es 0. Será impar cuando al ser dividido entre 2 su residuo es 1.

Sea **N = 123**

1. Debemos hallar el residuo de dividir N entre 2 (Utilizamos el operador *mod*) y lo almacenamos en R1.





R1 ← N *mod* 2 (R1 = 1)

2. Ahora verificamos a qué es igual el residuo. Si es igual a 0, será par. Si es igual a 1 (diferente de cero), será impar. (Utilizamos una estructura selectiva)

```
Si (R1 = 0) Entonces
    Escribir ("Es par")
Sino
    Escribir ("Es impar")
fin_si
```

Ejemplo: ***Diseñar un algoritmo que permita leer un número y que mediante un mensaje indique su paridad.*** **Solución:**

Pseudocódigo Paridad

Var *num, r*: entero

Inicio

Leer (*num*)

r ← *num* ***mod*** 2

Si (*r* = 0) **Entonces**

Escribir *(" Es par ")*

Sino

Escribir (" *Es impar* ")

fin_si

Fin

Podemos mejorar nuestro algoritmo si utilizamos solo una variable.

Pseudocódigo Paridad

Var *num*: entero

Inicio

Leer (*num*)

Si (*num* ***mod*** 2 = 0) **Entonces**

Escribir *(" Es par ")*

Sino

Escribir (" *Es impar* ")

fin_si

Fin

Ejemplo: ***Algoritmo para resolver una ecuación paramétrica.***

Nota: Recordar que una ecuación paramétrica tiene la forma: ax=b. Si a ≠ 0 entonces la solución es única y sería x=b/a; Si a = 0 y b = 0 entonces habrían infinitas soluciones; Si a = 0 y b ≠ 0 entonces no existiría solución. **Solución:**

```
Pseudocódigo Ecuación_Paramétrica
Var     a, b, x: real
Inicio
   Leer (a, b)
  Si (a ≠ 0) entonces
     x ← b/a
     Escribir ("solución única: ", x)
  sino
       Si (b=0) entonces
          Escribir ("Ecuación consistente indeterminada, infinitas soluciones")
       Sino
          Escribir ("Ecuación inconsistente, no hay solución")
       fin_si
  fin_si
Fin
```

Ejemplo: ***Dado un año, indicar si es bisiesto o no.***

Nota: Un año es bisiesto si es múltiplo de 4 y no de 100 o es múltiplo de 400.

Solución:

```
Pseudocódigo Año_Bisiesto
Var     año: entero
Inicio
   Leer (año)
  Si ((año mod 4 = 0 ∧ año mod 100 ≠ 0) ∨ año mod 400 = 0) entonces
        Escribir ("Es Bisiesto")
   sino
        Escribir ("No es bisiesto")
  fin_si
Fin
```

Sabemos que un número es múltiplo de otro cuando la división es exacta, por eso utilizamos el operador ***mod*** para calcular el residuo. Si este es cero sabremos que es múltiplo.

Ejemplo: ***Hallar el mayor de tres números reales.*** **Solución:**

Pseudocódigo Mayor

Var *a, b, c, máx:* entero

Inicio

Leer (*a, b, c*)

Si (a > b) **entonces**
máx ← *a*
sino
máx ← *b*
fin_si

Si (c > máx) **entonces**
máx ← *c*
fin_si

Escribir (*máx*)

Fin

Ejemplo: ***Diseñe un algoritmo que calcule el nuevo sueldo de un empleado conociendo su sueldo actual bajo el siguiente criterio. Aumente en 12% su sueldo si este es superior a s/.100 y un 20% en caso contrario. Solución***

Pseudocódigo Sueldo

Var *sueldo, nsueldo:* real

Inicio

Leer (*sueldo*)

Si (*sueldo* > 100) **entonces**
nsueldo ← *1.12 * sueldo*
sino
nsueldo ← *1.2 * sueldo*
fin_si

Escribir (*nsueldo)*

Fin

Ejemplo: ***Dado 3 números, determinar si la suma de cualquier pareja de ellas es igual al tercero. Si se cumple la condición, escribir "iguales" y en caso contrario "distintos".***

Solución: Vamos a resolverlo de dos formas:

Primera Forma:

Pseudocódigo Ejemplo13
Var *a, b, c:* entero
Inicio
Leer (*a, b, c*)
Si (*a+b=c*) **entonces**
Escribir ("Iguales")
sino
Si (b+c=a) **entonces**
Escribir ("*Iguales*")
sino
Si (a+c=b) **entonces**
Escribir ("*Iguales*")
sino
Escribir ("*Distintos*")
fin_si
fin_si
fin_si
Fin

Segunda Forma: Observen cómo el uso del operador lógico "$\vee$" disminuye considerablemente el número de estructuras selectivas dobles.

Pseudocódigo Ejemplo_13
Var *a, b, c:* entero
Inicio
Leer (*a, b, c*)
Si (*a+b=c* $\vee$ *a+c=b* $\vee$ *b+c=a)* **entonces**
Escribir *("Iguales")*
sino
Escribir ("*Distintos*")
fin_si
Fin

Ejemplo: ***Una empresa química paga a sus vendedores un sueldo básico quincenal de s/. 250 más un % del total de las ventas efectuadas según la siguiente tabla.***

Monto total de ventas	*% de pago*
MV ≥S/. 2000	*15%*
S/. 1500 ≤MV ≤S/. 2000	*12%*
S/. 0 ≤MV ≤S/. 1500	*10%*

Por otro lado si el sueldo del vendedor supera los S/. 800, éste se somete a un impuesto del 10% sobre su sueldo total. Desarrolle un algoritmo que permita conocer el monto del descuento por impuesto, el sueldo neto y el sueldo total.

Solución

Voy a utilizar las siguientes variables: *sb*: sueldo básico, *mtv*: monto total de las ventas, *desc*: descuento, *sn*: sueldo neto.

Pseudocódigo Sueldo

Var *sb, mtv, desc, sn:* real

Inicio

```
    sb ← 250.0
    Leer (mtv)
    Si (mtv ≥ 2000) entonces
        st ← sb + 0.15 * mtv
    sino
        Si (mtv ≥ 1500) entonces
            st ← sb + 0.12 * mtv
        sino
            st ← sb + 0.1 * mtv
        fin_si
    fin_si
    Si (st > 8000) entonces
        desc ← 0.1 * st
        sn ← st - desc
    sino
        desc ← 0
        sn ← st
    fin_si
    Escribir (st, desc, sn)
```

Fin

Este problema puede ser resuelto cambiando la última parte de la siguiente forma:

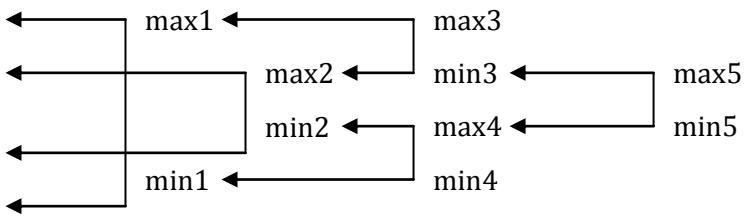
```
Pseudocódigo Sueldo
Var     sb, mtv, desc, sn: real
Inicio
   sb ← 250.0
   Leer (mtv)
  Si (mtv ≥ 2000) entonces
       st ← sb + 0.15 * mtv
   sino
       Si (mtv ≥ 1500) entonces
           st ← sb + 0.12 * mtv
       sino
           st ← sb + 0.1 * mtv
       fin_si
  fin_si
    Si (st > 8000) entonces
       desc ← 0.1 * st
    sino
       desc ← 0
    fin_si
   sn ← st - desc
   Escribir (st, desc, sn)
Fin
```

Ejemplo: ***Dado 3 números, presentarlos en forma decreciente.*** **Solución**

```
Pseudocódigo Ejemplo_15
Var     a, b, c: entero
Inicio
    Leer (a, b, c)
  Si (a > b) entonces
      Si (b>c) entonces
          Escribir (a, b, c)
      sino                (* c ≥ b *)
          Si (a>c) entonces
              Escribir (a, c, b)
          sino            (* c ≥ a *)
              Escribir (c, a, b)
          fin_si
      fin_si
  Sino                    (* b ≥ a *)
      Si (a>c) entonces
          Escribir (b, a, c)
      sino                (* c ≥ a *)
          Si (b>c) entonces
              Escribir (b, c, a)
          sino                (* c ≥ b *)
              Escribir (c, b, a)
          fin_si
      fin_si
  fin_si
Fin
```

Ejemplo: ***Dado 4 números enteros diferentes, presentar el segundo mayor.*** **Solución**

Para resolver este problema, vamos a tratar de ordenar los números de mayor a menor, y para eso realizaremos comparaciones de la siguiente forma:



Los números que van a ser ingresados, van a ser almacenados en las variables a, b, c y d. Luego comparamos "a" y "d" y tratamos de almacenar en "a" el mayor de entre estos dos, y almacenamos en "d" el menor de entre estos dos. Para eso usamos la siguiente instrucción:

```
Si (a < d) entonces
    aux ← a
    a ← d
    d ← aux
fin_si
```

Lo mismo haremos con b y c.

```
Si (b < c) entonces
    aux ← b
    b ← c
    c ← aux
fin_si
```

Entonces obtendremos dos máximos y dos mínimos parciales: max1, max2, min2, min1, los cuales estarán almacenados en ese orden en las variables a, b, c, d.

Luego compararemos los dos máximos parciales (max1 y max2 que están almacenados en a y b) de tal forma que tratemos de almacenar el mayor de entre estos dos en "a" y el menor de entre estos dos en "b". De manera análoga se comparará los dos mínimos parciales (min2 y min1 que están almacenados en c y d) de tal forma que tratemos de almacenar el menor de entre estos dos en "d" y el mayor de entre estos dos en "c". De esta forma estamos asegurando que el máximo de los 4 números iniciales se encuentre almacenado en "a" y que el mínimo de los 4 números se encuentre almacenado en "d". Por último queda comparar "b" y "c" que de tal forma que tratemos de almacenar en "b" el mayor de entre estos dos y en "c" el menor de entre estos dos. En conclusión, la finalidad de este método fue el de ordenar los números ingresados de mayor a menor pues de esta forma se sabe quién es el segundo mayor, tercero mayor, segundo menor, etc. (a > b > c > d)

Pseudocódigo Segundo_Mayor

Var *a, b, c, d, aux:* entero

Inicio

Leer (*a, b, c, d*)

Si (a < d) **entonces**
aux ← a
a ← d
d ← aux
fin_si

Si (b < c) **entonces**
aux ← b
b ← c
c ← aux
fin_si

Si (a < b) **entonces**
aux ← a
a ← b
b ← aux
fin_si

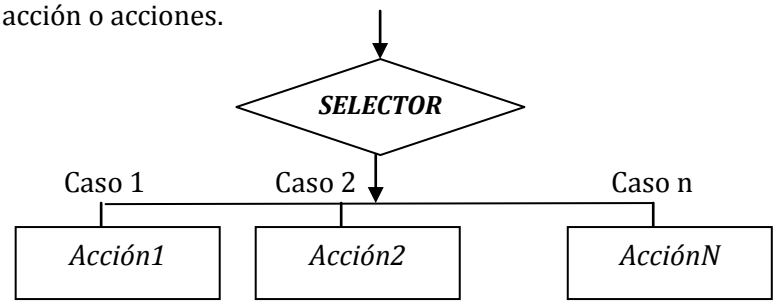
Si (c < d) **entonces**
aux ← c
c ← d
d ← aux
fin_si

Si (b < c) **entonces**
aux ← b
b ← c
c ← aux
fin_si

Escribir ("*El segundo mayor es: ",* b)

Fin

8.2.3. ESTRUCTURA SELECTIVA MÚLTIPLE: (***En_caso, sino, fin_caso***), Se utiliza cuando existan más de dos alternativas o elecciones posibles. Se evalúa un **Selector** que podrá tener n valores distintos: 1, 2, 3, … ,n y de acuerdo al valor que se obtenga se realizará o ejecutará alguna acción o acciones.



En_caso (*Selector*)

Caso1: Acción1
Caso2: Acción2
.
.
.
CasoN: AcciónN
Sino
Acción X
fin_caso

Donde:

SELECTOR: Es una variable del tipo ordinal, es decir, puede ser:

- ✓ Variable del tipo caracter (excepto los especiales)
- ✓ Variable del tipo numérico entero
- ✓ Variable del tipo lógico

No puede ser numérico real ni cadena de caracteres

CASO: Los casos son constantes del tipo ordinal (no pueden haber expresiones), vendrían a representar las alternativas y podrían tener las siguientes formas:

5

5, 6, 7

'A'

'A', 'B', 'C'

V (constante lógica verdadera)

F (constante lógica Falso)

1. . 5 (es equivalente a 1, 2 , 3, 4, 5)

'B'. . 'F' (es equivalente a B, C, D, E, F)

OBSERVACIONES:

- ✓ Como se puede apreciar, los **casos** o alternativas pueden estar conformados por un valor o por una lista de valores.
- ✓ Los valores que toman los **casos** no tienen porqué ser necesariamente consecutivos. **Ejemplo:**

  1, 3, 7, 12

- ✓ Además se puede apreciar que los **casos** pueden ser rangos de constantes numéricas o de caracteres. En general, para indicar un rango de constantes del tipo ordinal (caracteres, números enteros,) se tiene el siguiente formato:

  Lím_inf . . Lím_sup

- ✓ El uso del **Sino** es opcional.

Funcionamiento de la instrucción:

El **SELECTOR** se compara con cada uno de los casos, si hay coincidencia, realiza o se ejecuta la sentencia correspondiente y luego continúa con el resto del algoritmo. Si es que no hay coincidencia con ninguno de los casos y si:

Tenemos el SINO: Entonces se realiza la sentencia que se encuentra dentro de este bloque.
No tenemos el SINO: Entonces no realiza ninguna acción y la sentencia termina.

Ejemplo: ***Se desea determinar el salario mensual de un obrero conociendo el número total de horas trabajadas al mes y el turno en que labora. De acuerdo al turno en que labora, se tiene las siguientes tarifas.***

Turno	*Pago x Hora*
M	*30.5*
T	*27.9*
N	*36.5*

Solución:

Pseudocódigo Salario
Var *horas_trab*: entero
turno: caracter
salario: real
Inicio
Leer (*horas_trab, turno*)
En_caso (*turno*)
'M': *salario* ← *horas_trab* * 30.5
'T': *salario* ← *horas_trab* * 27.9
'N': *salario* ← *horas_trab* * 36.5
fin_caso
Escribir (*salario*)
Fin

Como se observa, no se utilizó el **SINO**.

Ejemplo: ***Diseñe un algoritmo que escriba los nombres de los días de la semana en función del valor de una variable día introducida por el teclado.*** **Solución*:***

Pseudocódigo Días

Var *día*: entero

Inicio

Leer (*día*)
En_caso (*día*)

1: **Escribir** ("*Lunes*")
2: **Escribir** ("*Martes*")
3: **Escribir** ("*Miércoles*")
4: **Escribir** ("*Jueves*")
5: **Escribir** ("*Viernes*")
6: **Escribir** ("*Sábado*")
7: **Escribir** ("*Domingo*")

fin_caso

Fin

Ejemplo: ***Solo empleando la instrucción selectiva múltiple, halle el mayor de 3 números reales.*** **Solución*:***

Pseudocódigo Mayor

Var *a, b, c, máx*: entero

Inicio

Leer (*a, b, c*)
En_caso (*a* > *b*)

V: *máx* ← *a*
F: *máx* ← *b*

fin_caso
En_caso (*c* > *máx*)

V: *máx* ← *c*

fin_caso

Fin

Ejemplo: ***Diseñar un algoritmo para determinar el costo del servicio de una cía fumigadora de terrenos hacia un agricultor que tiene una determinada extensión de terreno. La tarifa depende del tipo de fumigación:***

Tipo		
1	*Fumigación contra malas hierbas*	*S/. 10 Ha*
2	*Fumigación contra langostas*	*S/ 20 Ha*
3	*Fumigación contra gusanos*	*S/. 30 Ha*
4	*Fumigación contra todo lo anterior*	*S/. 50 Ha*

Si el área a fumigar es mayor a 500 Ha, tiene un descuento del 5%; además, si el total a pagar por el servicio es mayor a S/. 1500 , tiene un descuento del 10% sobre el exceso.
Solución*:*

Pseudocódigo Fumigación
Var *costo, Ha*: real
tip: entero
Inicio
Leer (*tip, Ha*)
En_caso (*tip*)
1: *costo* ← *Ha * 10*
2: *costo* ← *Ha * 20*
3: *costo* ← *Ha * 30*
4: *costo* ← *Ha * 50*
fin_caso
Si (*Ha >500*) **entonces**
costo ← *0.95 * costo*
fin_si
Si (*costo >1500*) **entonces**
costo ← *0.9 * costo*
fin_si
Escribir (*costo*)
Fin

Ejemplo: ***En una olimpiada de tiro al blanco se llega a un acuerdo para que el puntaje final obtenido sea calculado en base al puntaje original alcanzado en el tiro (del 1 al 10) multiplicado por un factor, según la siguiente tabla:***

Puntaje Original	*Factor*
0	*0*
1 – 3	*3*
4 – 6	*8*
7 - 10	*10*

Se pide diseñar un algoritmo que calcule el puntaje final de un determinado jugador

Solución:

Pseudocódigo Juego

Var *punt, punt_final*: entero

Inicio

Leer (*punt*)

En_caso (*punt*)

0: *punt_final* ← *punt* * 0
1 . . 3: *punt_final* ← *punt* * 3
4 . . 6: *punt_final* ← *punt* * 8
7 . . 10: *punt_final* ← *punt* * 10

fin_caso

Escribir (*punt_final*)

Fin

Ejemplo: ***Se desea leer por teclado un número comprendido desde 1 al 10 y se desea visualizar un mensaje en la pantalla indicando su el número ingresado es par o impar (Use la instrucción selectiva múltiple) Solución:***

Pseudocódigo Número

Var *num*: entero

Inicio

Leer (*num*)

En_caso (*num*)
1, 3 , 5, 7, 9: **Escribir** (*"impar"*)
2, 4, 6, 8, 10: **Escribir** (*"par"*)
fin_caso

Fin

LA INSTRUCCIÓN CONTROVERSIAL: GOTO

En los inicios de la programación, era frecuente el uso de una instrucción de **transferencia incondicional** (salto) llamada **GOTO.** Esta instrucción, si bien es cierto, permitía simplificar algunos problemas, en muchas ocasiones su uso indiscriminado conducía a programas difíciles de comprender y mantener. (Para entender la lógica de los programas en donde se usaba la instrucción **GOTO** había necesidad de hacer engorrosos seguimientos en cada salto dentro de los bloques de código).

El Teorema de la Programación Estructurada fue el punto de partida de la controversia generada respecto al uso de la instrucción **GOTO.** Algunos programadores empezaron a criticar el uso de esta instrucción considerándola **dañina** y desaconsejable. La crítica más famosa que se conoce fue escrita por **Edsger Dijkstra** la cual fue una carta con título "Instrucción Go to considerada dañina" en el año 1968.

8.3. ESTRUCTURAS REPETITIVAS

Conocida también como: ESTRUCTURAS CÍCLICAS o ITERATIVAS.

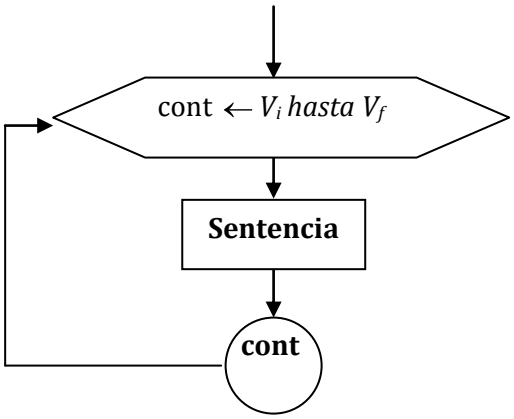
Las estructuras repetitivas son aquellas en las cuales se utilizan a las **estructuras de control repetitivas**, los cuales son también llamados bucles de repetición.

Existen tres estructuras repetitivas:

- ✓ Estructura Repetitiva **Desde** (Bucle con número de repeticiones preestablecido)
- ✓ Estructura Repetitiva **Mientras** (Bucle con entrada Controlada)
- ✓ Estructura Repetitiva **Repetir** (Bucle con salida controlada)

8.3.1. EL BUCLE CON NÚMERO DE REPETICIONES PREESTABLECIDO: (ESTRUCTURA REPETITIVA DESDE)

Es utilizado cuando el número de repeticiones o iteraciones se conoce, es decir, cuando ya sabemos el número de veces que vamos a repetir una misma acción. Por ejemplo: Para calcular el factorial de un número, sabemos el número de veces que se va a multiplicar el número por sus menores consecutivos hasta el 1.



Desde *cont* ← V_i *hasta* V_f
Instrucciones
fin_desde

Donde:

V_i: Valor inicial (no necesariamente empieza en uno)

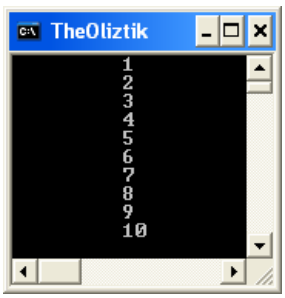
V_f: Valor final

V_i, V_f, cont: son del tipo ORDINAL

Funcionamiento: La sentencia se repite desde que **cont** toma el valor inicial V_i hasta que alcanza en forma secuencial el valor final V_f.

Por ejemplo:

```
Desde i ← 1 hasta 10
    Escribir ( i )
fin_desde
```




```
Desde car ← 'X' hasta 'Z'
    Escribir ( car )
fin_desde
```

En este curso, generalmente, solo se va a considerar avanzar de 1 en 1.

Ejemplo: ***Diseñar un algoritmo para calcular el factorial de un número entero positivo.***

Solución: Vamos a utilizar el contador "i".

Pseudocódigo Factorial

Var *i, num, fac*: entero

Inicio

Leer (*num*)

fac ← 1

desde *i* ← 1 *hasta* num

fac ← *fac* * i

fin_desde

Escribir (*fac*)

Fin

Ejemplo: ***Se pide sumar los n primeros números naturales.***

Solución: Vamos a utilizar un acumulador "S" y un contador "i". Recuerde que para el uso de un acumulador, en primer lugar hay que inicializarlo, es decir, asignarle un valor inicial. Generalmente los acumuladores se inicializan en cero.

```
Pseudocódigo Suma
Var     i, S, n: entero
Inicio
   Leer (n)
   S ← 0
   desde i ← 1 hasta n
      S ← S + i
   fin_desde
   Escribir (S)
Fin
```

FACTORIAL, DIVISORES, NÚMEROS PRIMOS

Ejemplo: ***Diseñar un algoritmo para calcular el factorial de n números naturales leídos.***

Solución: Vamos a utilizar dos instrucciones repetitivas, una para calcular el factorial de un número (tal y como lo hicimos en el ejercicio 23, y otra instrucción que permita la lectura (ingreso) de los "n" números. Además, debemos usar dos contadores diferentes: "i" y "j", uno para cada estructura repetitiva.

```
Pseudocódigo Factorial
Var     i, j, num, n, fac: entero
Inicio
   Leer (n)    (* n es la cantidad de números que se va a ingresar *)
   desde i ← 1 hasta n
      Leer (num)
      fac ← 1
      desde j ← 1 hasta num
         fac ← fac * j
      fin_desde
      Escribir (fac)
   fin_desde
Fin
```

Ejemplo: ***Diseñe un algoritmo tal que, dado un número entero positivo, presente la cantidad de divisores.***

Solución: Aquí vamos a usar el operador ***mod***, debido a que si un número es divisible por otro, entonces la división de ambos debe ser exacto y el residuo debe ser cero. Usaremos una estructra repetitiva dentro de la que se encontrará una estructura selectiva, en donde la condición será: "si la división de un número entre otro arroja como residuo cero, es decir, **num1** ***mod*** **num2 = 0**, entonces se contabiliza, lo cual se realiza a partir de un contador que representa la cantidad de divisores, **cont** ← **cont+1** (sin olvidar de inicializar el contador con el valor de cero).

```
Pseudocódigo Cantidad_Divisores
Var     cont, i, num: entero
Inicio
    cont ← 0
    Leer (num)
    desde i ← 1 hasta num
        Si (num mod i = 0) entonces
        cont ← cont +1
        fin_si
    fin_desde
    Escribir (cont)
Fin
```

Ejemplo: ***Diseñe un algoritmo tal que, dado un número entero positivo, presente todos sus divisores y la cantidad de estos.***

Solución: Similar al ejercicio anterior.

```
Pseudocódigo Divisores
Var     cont, i, num: entero
Inicio
    cont ← 0
    Leer (num)
  desde i ← 1 hasta num
      Si (num mod i = 0) entonces
          Escribir ( i )
          cont ← cont +1
      fin_si
  fin_desde
    Escribir (cont)
Fin
```

Ejemplo: ***Diseñe un algoritmo tal que, para un grupo de n números enteros positivos, presente todos sus divisores y la cantidad de estos.***

Solución: Al ejercicio anterior se le agrega una estructura repetitiva.

```
Pseudocódigo Divisores
Var     cont, i, j, num, n: entero
Inicio
    Leer (n)
  desde i ← 1 hasta n
      cont ← 0
      Leer (num)
    desde j ← 1 hasta num
        Si (num mod j = 0) entonces
            Escribir ( j )
            cont ← cont +1
        fin_si
    fin_desde
      Escribir (cont)
  fin_desde
Fin
```

Ejemplo: ***Diseñe un algoritmo tal que dado un número entero positivo, identifique si es primo.***

Solución: Un número es primo cuando tiene como divisores únicamente a sí mismo y a la unidad. Para identificar si un número es primo, lo que se hace es contar todos sus divisores, y solo cuando el número de divisores es dos, será primo.

```
Pseudocódigo Primo
Var      i, num, cont: entero
Inicio
    cont ← 0
    Leer (num)
    desde i ← 1 hasta num
        Si (num mod i = 0) entonces
            cont ← cont +1
        fin_si
    fin_desde
    Si (cont=2) entonces
        Escribir ("Es primo")
    sino
        Escribir ("No es primo")
    fin_si
Fin
```

Ejemplo: ***Para un conjunto de "n" números enteros positivos, identifique cuando es primo, o compuesto mediante un mensaje en la pantalla: "es primo" o "es compuesto". Si es compuesto, presente sus divisores y la cantidad de divisores que posee. Además, al final presente cuantos de los n números ingresados resultaron ser primos y cuántos resultaron ser compuestos.***

Solución: Este ejercicio es una combinación de los ejemplos anteriores, pero, aquí a parte de utilizar un contador para la cantidad de divisores (cont_d) de cada número ingresado, vamos a utilizar dos contadores adicionales para contabilizar la cantidad de primos (cont_p) y la cantidad de compuestos (cont_c).

```
Pseudocódigo Ejemplo
Var     n, num, i, j, cont_d, cont_p, cont_c: entero
Inicio
    cont_p ← 0
    cont_c ← 0
    Leer (n)
  desde i ← 1 hasta n
      Leer (num)
    Si (num = 1) entonces
        Escribir ("no es primo ni compuesto, es un número simple")
    sino
        cont_d ← 0
      desde j ← 1 hasta num
          Si (num mod j = 0) entonces
              Escribir ( j )
              cont_d ← cont_d + 1
          fin_si
      fin_desde
      Si (cont = 2 ) entonces
              Escribir ("es primo")
              cont_p ← cont_p + 1
        sino
              Escribir ("Es compuesto")
              Escribir ("Número de divisores: ", cont_d)
              cont_c ← cont_c + 1
      fin_si
    fin_si
  fin_desde
    Escribir ("Cantidad de números primos: ", cont_p)
    Escribir ("Cantidad de números compuestos: ", cont_c)
Fin
```

Ejemplo: ***La ASJ está interesada en promover la natación y para ello desea conocer jóvenes con las siguientes condiciones: edad (menor de 18 años), estatura (mínimo 1.70 cm.) y peso (máximo 70 Kg.). Diseñar un algoritmo en el cual se lean la edad, estatura, peso de un grupo de n deportistas y verifique cuantos de ellos cumplen las condiciones impuestos. Presentar la edad, peso, estatura el número de jóvenes que cumplen y el peso promedio de estos.***

Solución:

Pseudocódigo ASJ

Var *edad, i, n, cont*: entero
talla, peso, peso_prom, suma_p: real

Inicio

cont ← 0
suma_p ← 0
Leer (*n*)
Desde *i* ← 1 *hasta n*
Leer (*edad, talla, peso*)
Si (*edad* < 18 ∧ *talla* ≥ 1.7 ∧ *peso* ≤ 70) **entonces**
cont ← *cont* + 1
suma_p ← *suma_p* + *peso*
Escribir *(edad, talla, peso)*
fin_si
fin_desde
Si *(cont* ≠ *0)* **entonces**
peso_prom ← *suma_p* / *cont*
Escribir *(peso_prom)*
Sino
Escribir *("Ninguno cumple las condiciones")*
fin_si

Fin

MÁXIMOS Y MÍNIMOS

Ejemplo: ***Para n estudiantes se desea leer las notas (desde 0 hasta 20) y se pide averiguar la mayor nota.***

Solución: Cuando se conoce el rango de valores que puede asumir una variable que represente a un valor **máximo**, debemos inicializarlo con el **menor** de los valores del rango. Por ejemplo, para este problema se van a leer notas los cuales se encuentran en el rango de 0 a 20, luego mediante una serie de instrucciones vamos tratar de hallar cual fue la **mayor** de todas las notas ingresadas la cual deberá ser almacenada en una variable que llamaremos ***maxnota***, pero antes de todo, debemos inicializar a nuestra variable ***maxnota*** con el valor cero (que es el **menor** del rango 0 – 20).

Pseudocódigo Ejemplo

```
Var     n, nota, maxnota, i: entero
Inicio
  Leer (n)
  maxnota ← 0
  Desde i ← 1 hasta n
    Leer (nota)
    Si (maxnota < nota) entonces
      maxnota ← nota
    fin_si
  fin_desde
  Escribir (maxnota)
Fin
```

Estamos inicializando la variable entera *maxnota* con el valor 0 (cero), que es el menor de los valores del rango (0 a 20)

Ejemplo: ***Para n estudiantes se desea leer las notas (Desde 0 hasta 20 y se pide averiguar la menor nota.***

Solución: De forma análoga al ejercicio anterior. Cuando se conoce el rango de valores que puede asumir una variable que represente a un valor **mínimo**, debemos inicializarlo con el **mayor** de los valores del rango. Por ejemplo, para este problema se van a leer notas los cuales se encuentran en el rango de 0 a 20, luego mediante una serie de instrucciones vamos tratar de hallar cual fue la **menor** de todas las notas ingresadas la cual deberá ser almacenada en una variable que llamaremos ***minnota***, pero antes de todo, debemos inicializar a nuestra variable ***minnota*** con el valor 20 (que es el **mayor** del rango 0 – 20).

Pseudocódigo Ejemplo

```
Var     n, nota, minnota, i: entero
Inicio
  Leer (n)
  minnota ← 20
  Desde i ← 1 hasta n
    Leer (nota)
    Si (minnota > nota) entonces
      minnota ← nota
    fin_si
  fin_desde
  Escribir (minnota)
Fin
```

Estamos inicializando la variable entera *minnota* con el valor 20 (veinte), que es el mayor de los valores del rango (0 a 20)

Ejemplo: ***Para n estudiantes se desea leer los códigos y se pide averiguar el mayor código ingresado.***

Solución: Cuando **no** se conoce el rango de valores que puede asumir una variable que represente a un valor **máximo**, debemos inicializarlo con el **primero** de los datos ingresados. Por ejemplo, para este problema se van leer (ingresar) códigos de los que no sabemos en qué rango se encuentran, y mediante una serie de instrucciones vamos tratar de hallar cual fue el mayor de todos los códigos ingresados el cual deberá ser almacenado en una variable que llamaremos ***maxcod***, pero antes de todo, debemos inicializar a nuestra variable ***maxcod*** con el **primero** de los datos ingresados. (En el supuesto caso en el que nos hubieran dicho que los códigos se encuentran en el rango 100 – 999 , entonces hubiéramos inicializado a la variable ***maxcod*** con 100)

Se puede resolver de dos formas que no difieren mucho.

Pseudocódigo Ejemplo

```
Var     n, cod, maxcod, i: entero
Inicio
   Leer (n)
   Desde i ← 1 hasta n
      Leer (cod)
      Si (i = 1) entonces
         maxcod ← cod
      sino
         Si (maxcod < cod) entonces
            maxcod ← cod
         fin_si
      fin_si
   fin_desde
   Escribir (maxcod)
Fin
```

Estamos inicializando la variable entera *maxcod* con el primero de los datos ingresados (cuando i = 1, es decir, a la primera vez que se ejecuta la instrucción **desde**)

Cuando i = 2 , se lee el segundo código, y se compara el código leído anteriormente (que fue almacenado en *maxcod*) con el código leído ahora (que a sido almacenado en *cod*), de tal forma que *maxcod* se quede con el mayor valor de entre estos dos. Así se va repitiendo el bucle y nos aseguramos que *maxcod* almacene el mayor de todos los códigos leídos.

```
Pseudocódigo Ejemplo
Var      n, cod, maxcod, i: entero
Inicio
  Leer (n)
  Leer (cod)
  maxcod ← cod
  Desde i ← 2 hasta n
    Leer (cod)
    Si (maxcod < cod) entonces
      maxcod ← cod
    fin_si
  fin_desde
  Escribir (maxcod)
Fin
```

Estamos inicializando la variable entera *maxcod* con el primero de los datos ingresados

Debido a que inicializamos la variable *maxcod* fuera de la instrucción repetitiva **desde** y el total de datos ingresados debe de ser n, entonces, la instrucción repetitiva solo se repetirá n-1 veces (desde i = 2 hasta n)

Ejemplo: ***Para n estudiantes se desea leer las notas (desde 0 hasta 20) y se pide averiguar la mayor nota y cuántos lo obtuvieron.*** **Solución*:***

Pseudocódigo Ejemplo

Var *n, nota, maxnota, i, cont:* entero

Inicio

Leer (*n*)

cont ← 0

maxnota ← 0

desde i ← 1 *hasta* n

Leer (*nota*)

Si (*maxnota* < *nota*) **entonces**

maxnota ← *nota*

cont ← 1

Sino

Si (*maxnota* = *nota*) **entonces**

cont ← *cont* + 1

fin_si

fin_si

fin_desde

Escribir (*maxnota, cont*)

Fin

Como vamos a utilizar un contador, no olvidar inicializarlo en cero

Debido a que se conoce el rango, inicializamos la variable entera *maxnota* con el valor de cero

En esta parte se va a encontrar la mayor nota ingresada, además de actualizar el contador a 1. Actualizar el contador a 1 quiere decir, regresar el valor de este a 1, porque puede darse el caso en donde se haya estado contabilizando una nota que parecía que era la mayor de todas, y luego cuando se ingrese una nota mayor, el contador debe volver a 1.

En esta parte del algoritmo, se va a contabilizar el número de coincidencias con la mayor nota.

PRUEBA DE ESCRITORIO: Llamaremos prueba de escritorio al proceso de comprobación de un algoritmo para ver si funciona correctamente. Para realizar esta prueba se asumen ciertos valores de entrada para luego seguir las acciones descritas en el algoritmo hasta el final del mismo. Una vez terminado este proceso, se procede a verificar los resultados. Si los resultados son erróneos, tenemos que corregir el algoritmo y hacer los ajustes necesarios.

Para probar si funciona el algoritmo de este ejercicio, supongamos que las notas que se debieron ingresar fueron: 05, 10, 15, 15, 12, 17, 17, 05. El proceso sería el siguiente:

Debemos ingresar la cantidad de notas que vamos a almacenar, en este caso n debe ser 8.
Leer (*n*)
n = 8

Inicializamos el *contador* con el valor de cero y a la variable *maxnota* también con cero.
cont ← 0
maxnota ← 0

Comenzamos con la instrucción repetitiva DESDE
Desde i ← 1 *hasta* n

*** CUANDO i = 1 :**
Ingresamos la primera nota (la cual será leída por la computadora)
Leer (*nota*)
nota = 05

Comparamos la variable *maxnota* que almacena el valor de 0 con la nota ahora ingresada que almacena el valor de 05

Si (*maxnota* < *nota*) **entonces**

Debido a que sí se cumple la condición (00 < 05) almacenamos en la variable maxnota, el valor de nota (05)

maxnota ← *nota*
maxnota = 05
Y el contador lo inicializamos en 1 para indicar que encontramos una mayor nota.
cont ← 1

*** CUANDO i = 2 :**
Ingresamos la segunda nota
Leer (*nota*)
nota = 10

Comparamos la variable *maxnota* que almacena el valor de 05 con la nota ahora ingresada que almacena el valor de 10

Si (*maxnota* < *nota*) **entonces**

Debido a que sí se cumple la condición (05 < 10) almacenamos en la variable *maxnota*, el valor de nota (10)

maxnota ← *nota*
maxnota = 10
Y el contador lo inicializamos en 1 (lo actualizamos a 1) para indicar que encontramos una mayor nota (mayor que la anterior)
cont ← 1

*** CUANDO i = 3:**
Ingresamos la tercera nota
Leer (*nota*)
nota = 15

Comparamos la variable maxnota que almacena el valor de 10 con la nota ahora ingresada que almacena el valor de 15

Si (*maxnota* < *nota*) **entonces**

Debido a que sí se cumple la condición (10 < 15) almacenamos en la variable maxnota, el valor de nota (15)

maxnota ← *nota*
maxnota = 15
Y el contador lo inicializamos en 1 (lo actualizamos a 1) para indicar que encontramos una mayor nota (mayor que la anterior)
cont ← 1

*** CUANDO i = 4:**
Ingresamos la cuarta nota
Leer (*nota*)
nota = 15

Comparamos la variable *maxnota* que almacena el valor de 10 con la nota ahora ingresada que almacena el valor de 15

Si (*maxnota* < *nota*) **entonces**

Debido a que no se cumple la condición (15 < 15), nos vamos hacia el **Sino**

Sino
Evaluamos la condición que se encuentra dentro del **Sino**
Si (*maxnota* = *nota*) **entonces**

Debido a que sí se cumple esta condición (15 = 15), ejecutamos lo que sigue a continuación
cont ← *cont* + 1
Contabilizamos las coincidencias con esta supuesta mayor nota (por lo menos hasta ahora)
cont = 1 + 1 = 2

*** CUANDO i = 5:**
Ingresamos la quinta nota
Leer (*nota*)
nota = 12

Comparamos la variable *maxnota* que almacena el valor de 15 con la nota ahora ingresada que almacena el valor de 12

Si (*maxnota* < *nota*) **entonces**

Debido a que no se cumple la condición (15< 12), nos vamos hacia el **Sino**

Sino
Evaluamos la condición que se encuentra dentro del **Sino**

Si (*maxnota* = *nota*) **entonces**
Debido a que tampoco se cumple esta condición (15 = 12), nos salimos de la instrucción condicional y pasamos a la siguiente repetición.

*** CUANDO i = 6:**
Ingresamos la sexta nota
Leer (*nota*)
nota = 17

Comparamos la variable *maxnota* que almacena el valor de 15 con la nota ahora ingresada que almacena el valor de 17

Si (*maxnota* < *nota*) **entonces**

Debido a que sí se cumple la condición (15 < 17) almacenamos en la variable *maxnota*, el valor de nota (17)

maxnota ← *nota*
maxnota = 17
Y el contador lo inicializamos en 1 (lo actualizamos a 1) para indicar que encontramos una mayor nota (mayor que la anterior)
cont ← 1
cont = 1

*** CUANDO i = 7:**
Ingresamos la séptima nota
Leer (*nota*)
nota = 17

Comparamos la variable *maxnota* que almacena el valor de 17 con la nota ahora ingresada que almacena el valor de 17

Si (*maxnota* < *nota*) **entonces**

Debido a que no se cumple la condición (17 < 15), nos vamos hacia el **Sino**

Sino
Evaluamos la condición que se encuentra dentro del **Sino**

Si (*maxnota* = *nota*) **entonces**

Debido a que sí se cumple esta condición (17 = 17), ejecutamos lo que sigue a continuación
cont ← *cont* + 1
Contabilizamos las coincidencias con esta supuesta mayor nota (por lo menos hasta ahora)
cont = 1 + 1 = 2

*** CUANDO i = 8:**

Ingresamos la octava nota
Leer (*nota*)
nota = 05

Comparamos la variable *maxnota* que almacena el valor de 17 con la nota ahora ingresada que almacena el valor de 05

Si (*maxnota* < *nota*) **entonces**
Debido a que no se cumple la condición (17< 05), nos vamos hacia el **Sino**

Sino
Evaluamos la condición que se encuentra dentro del **Sino**

Si (*maxnota* = *nota*) **entonces**

Debido a que tampoco se cumple esta condición (17 = 05), nos salimos de la instrucción condicional.

Ahora que ya se terminó número de notas a ingresar, *maxnota* almacena la mayor de todas las notas ingresadas que fue 17. Solo nos queda mostrarlo por la pantalla y la cantidad de veces que se fue ingresado
Escribir (*maxnota, cont*)

En la pantalla deberá mostrarse “17” y “2”

Ejemplo: ***Para n estudiantes se desea leer las notas (desde 0 hasta 20) y se pide averiguar la menor nota y cuántos lo obtuvieron. Solución:***

```
Pseudocódigo Ejemplo
Var      n, nota, minnota, i, cont: entero
Inicio
  Leer (n)
  cont ← 0
  minnota ← 20
  desde i ← 1 hasta n
    Leer (nota)
    Si (minnota > nota) entonces
      minnota ← nota
      cont ← 1
    Sino
      Si (minnota = nota) entonces
          cont ← cont + 1
      fin_si
    fin_si
  fin_desde
  Escribir (minnota, cont)
Fin
```

Inicializar con el valor numérico cero al contador

Debido a que se conoce el rango de valores de notas (0-20), inicializamos la variable entera *minota* con el valor máximo del rango, es decir, veinte.

Este algoritmo se puede comprobar mediante una **Prueba de Escritorio** similar a la del ejercicio anterior.

Ejemplo: ***Para un grupo de n alumnos se desea leer el código (entero de 3 dígitos), nota (de 0 a 20) y sexo (M o F), luego se pide lo siguiente:***

a) Presentar la menor nota en los hombres y el mayor de los códigos que la tiene.

b) Presentar el número de alumnos hombres que tiene la menor nota.

Solución:

```
Pseudocódigo Ejemplo
Var     cod, n, maxcod, i, cont: entero
        sexo: caracter
        nota, minnota: real
Inicio
  Leer (n)
  maxcod ← 100
  minnota ← 20
  cont ← 0
  desde i ← 1 hasta n
      Leer (cod, nota, sexo)
    Si (sexo = 'M') entonces
        Si (minnota > nota) entonces
            minnota ← nota
            maxcod ← cod
            cont ← 1
        Sino
            Si (minnota = nota) entonces
                cont ← cont + 1
                Si (maxcod < cod ) entonces
                    maxcod ← cod
                fin_si
            fin_si
        fin_si
    fin_si
  fin_desde
  Escribir (minnota, maxcod, cont)
Fin
```

Ejemplo: ***Para un conjunto de n personas se desea averiguar la edad promedio por sexo y cuál es la edad mayor en cada caso (la máxima edad en cada sexo). Desarrolla un algoritmos que reciba las edades de las n personas y entregue la información pedida.***

Pseudocódigo Ejemplo

Var *edad, n, i , contM, contF, maxM, maxF, sumaM, sumaF*: entero
sexo: caracter

Inicio

Leer (*n*)
contM ← 0
contF ← 0
sumaM ← 0
sumaF ← 0
maxM ← 0
maxF ← 0

desde i ← 1 *hasta* n
Leer (*edad, sexo*)
En_caso (*sexo*)
'M', 'm': *sumaM* ← *sumaM + edad*
contM ← *contM* + 1
Si (*edad > maxM*) **entonces**
maxM ← *edad*
fin_si
'F', 'f': *sumaF* ← *sumaF + edad*
contF ← *contF* + 1
Si (*edad > maxF*) **entonces**
maxF ← *edad*
fin_si
fin_caso
fin_desde
Escribir (*maxM, maxF*)
Si (*contM* = 0) **entonces**
Escribir ("*no hay promedio*")
Sino
Escribir (*sumaM* ***div*** *contM*) **(* Para mostrar un promedio entero *)**
fin_si

```
Si (contF = 0) entonces
    Escribir ("no hay promedio")
sino
    Escribir (sumaM div contF)      (* Para mostrar un promedio entero *)
fin_si
Fin
```

Ejemplo: **SEGUNDA MAYOR EDAD.** ***Para un conjunto de n personas, se desea leer sus edades y luego se desea presentar la segunda mayor edad.*** **Solución;**

Pseudocódigo Segunda_mayor_edad

Var *edad, i, edad, max1, max2:* entero

Inicio

Leer (*n*)

max1 ← 0

max2 ← 0

desde i ← 1 *hasta* n

Leer (*edad*)

Si (*i* = 1) **entonces**

Si (*max1* < *edad*) **entonces**

max1 ← *edad*

fin_si

Sino **(* cuando i = 2, 3, 4, ... , n *)**

Si (*max1* < *edad*) **entonces**

max2 ← *max1*

max1 ← *edad*

sino **(* no considero la igualdad edad = max1 *)**

Si (*max1* > *edad*) **entonces**

Si (*max2* < *edad*) **entonces**

max2 ← *edad*

fin_si

fin_si

fin_si

fin_si

fin_desde

Si (*max2* = 0) **entonces** **(*max2 será cero** si es que todos los datos ingresados **son iguales*)**

Escribir (*"No hay segunda mayor"*)

sino

Escribir (*max2*)

fin_si

Fin

Ejemplo: SEGUNDA MENOR EDAD. ***Para un conjunto de n personas, se desea leer sus edades y luego se desea presentar la segunda menor edad.***

Solución: A diferencia del ejercicio anterior, no vamos a inicializar las variables min1 y min2 en cero, sino con la primera edad leída.

```
Pseudocódigo Segunda_menor_edad
Var     edad, i, edad, min1, min2: entero
Inicio
  Leer (n)
 desde i ← 1 hasta n
    Leer (edad)
    Si (i = 1) entonces
        min1 ← edad
        min2 ← edad
    Sino      ( * i = 2, 3, 4, ... , n * )
        Si (min1 > edad) entonces
            min2 ← min1
            min1 ← edad
        sino
            Si (min1 < edad) entonces
                Si (min2 > edad) entonces
                  min2 ← edad
                fin_si
            fin_si
        fin_si
    fin_si
 fin_desde
 Si (min1 ≠ min2) entonces
    Escribir ("Segunda menor edad:", min2)
  sino
    Escribir ("No existe segunda menor edad")
 fin_si
Fin
```

Ejemplo: ***Para un conjunto de n datos reales se desea determinar el mayor de los datos negativos y cuántas veces aparece. Por ejemplo: para n=7, datos: 4, -7, 6, -3, 5, -3, 1***

Solución

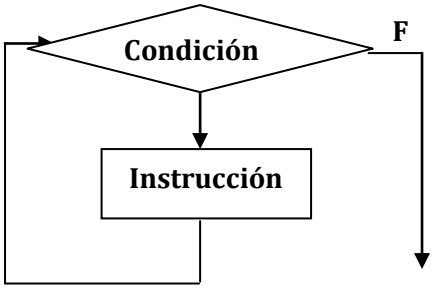
```
Pseudocódigo Mayor_negativo
Var     n, i, cont: entero
        num: real
        encont: lógico
Inicio
  encont ← F          <-- Vamos a inicializar a la variable centinela o variable switch con el valor lógico falso
  Leer (n)
  desde i ← 1 hasta n
    Leer (num)
    Si (num < 0) entonces          <-- Vamos a descartar los datos no negativos que sean ingresados
      Si ( ¬ encont) entonces
        maxneg ← num
        cont ← 1
        encont ← V
      sino
        Si (maxneg < num) entonces
          maxneg ← num
          cont ← 1
        sino
          Si (maxneg = num) entonces
            cont ← cont + 1
          fin_si
        fin_si
      fin_si
    fin_si
  fin_desde
  Escribir (maxneg, cont)
Fin
```

Explicación del ejemplo: Vamos a crear una variable llamada *maxneg* y esta la tenemos que inicializar con el primer dato negativo ingresado. Pero puede darse el caso que el primer dato ingresado no sea negativo, sino cero o positivo, ¿cómo sabremos en qué momento vamos a tener un dato negativo con el cual debemos inicializar nuestra variable *maxneg*? Para esto, vamos a utilizar una **variable switch** (*conocida también como bandera, centinela, flag, conmutador, etc*), el identificador para este será *encont,* será de tipo lógico y servirá para poder saber en qué momento se ingresará un dato negativo, de esta forma podremos inicializar la variable *maxneg* y podremos compararlo con los datos negativos siguientes (los datos no negativos ingresados no serán tomados en cuenta)

8.3.2 BUCLE CON ENTRADA CONTROLADA (ESTRUCTURA REPETITIVA MIENTRAS). ESTRUCTURA CON PRE-TEST DE PRUEBA

Es utilizada cuando el número de repeticiones o iteraciones no se conoce, aunque también sirve cuando sí se conoce el número de repeticiones. En esta estructura, el cuerpo del bucle se repite mientras se cumpla una determinada condición.

Funcionamiento: Cuando se ejecuta la instrucción mientras, lo primero que sucede es que se evalúa una condición. Si la condición resulta ser verdadera, se ejecuta el cuerpo del bucle, después se evalúa de nuevo la condición. Este proceso se repite una y otra vez mientras la condición sea verdadera. Cuando la condición resulte ser falsa, no se realiza ninguna acción que se encuentra dentro del cuerpo del bucle, sino que el algoritmo sigue con las demás instrucciones.




```
Mientras (condición) Hacer
    Instrucción
fin_mientras
```

OBSERVACIÓN

Esta instrucción tiene desde 0 hasta más repeticiones.

La condición debe modificarse dentro del bucle para evitar que se genere un bucle infinito.

Ejemplo: Un bucle finito.

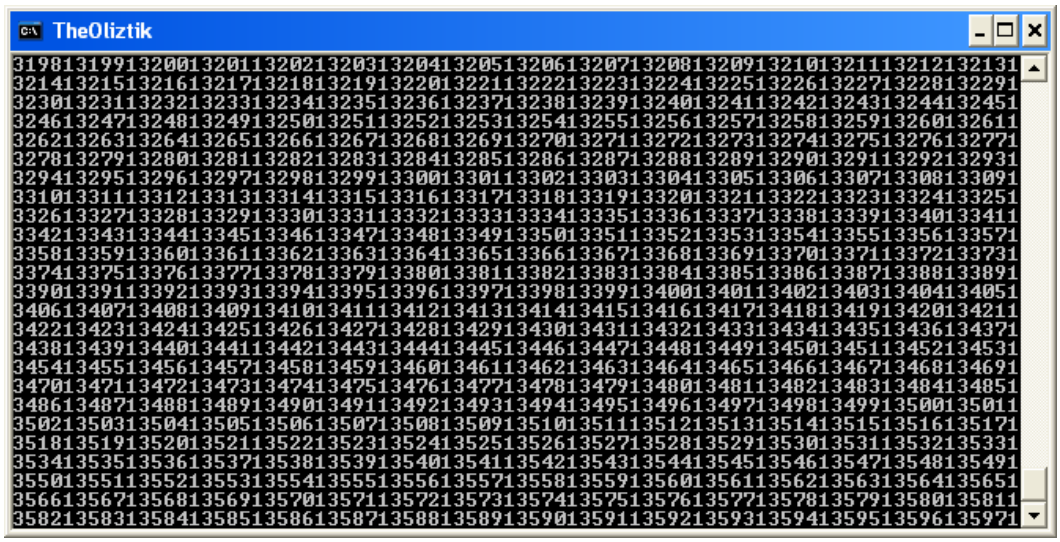
```
x ← 3
Mientras (x > 0) hacer
    Escribir (x)
    x ← x-1
fin_mientras
```

*En la pantalla se verá únicamente los valores positivos de x: 3, 2 y 1(según la condición), pues cuando x a toma un valor no positivo, la instrucción repetitiva "**Mientras**" termina*

Ejemplo: Un bucle infinito

```
x←1
Mientras (x > 0) hacer
    Escribir (x)
    x ← x+1
fin_mientras
```

En la pantalla se verán los valores de x: 2, 3, 4, …., etc. Y esto continuará hasta que la computadora se *"cuelgue"*, ya que la instrucción **Mientras** no terminará mientras x no tome un valor negativo lo cual nunca pasará pues x va en aumento



Ejemplo: ***Imprimir todos los números primos entre 2 y 100 (incluye a 2 y a 100), usando la instrucción mientras.***

Solución. Ya sabemos el algoritmo para identificar a un número primo. Ahora le agregaremos una estructura adicional.

Pseudocódigo Primos

Var *num, cont_div, i:* entero

Inicio

num ← *2*

Mientras (*num* ≤ 100) **hacer**

desde i ← *1 hasta* num

Si (*num* ***mod*** *i* = 0) **entonces**

cont_div ← *cont_div+1*

fin_si

fin_desde

Si (*cont_div = 2*) **entonces**

Escribir (*num*)

fin_si

num ← *num+1* ← La variable *num* debe ir aumentando de uno en uno

fin_mientras

Fin

El ejemplo también pudo haberse resuelto con el uso de dos estructuras repetitivas **desde** de la siguiente forma:

Pseudocódigo Primos

Var *num, cont_div, i:* entero

Inicio

desde *num* ← 2 *hasta* 100
  desde i ← *1 hasta* num
    Si (*num* ***mod*** *i* = 0) **entonces**
      cont_div ← *cont_div+1*
    fin_si
  fin_desde
  Si (*cont_div = 2*) **entonces**
    Escribir (*num*)
  fin_si
fin_desde

Fin

Usando la instrucción **desde** no hubo la necesidad de ir aumentando la variable *num*, porque esta aumenta una unidad automáticamente durante cada repetición

Ejemplo: ***Dado un número entero positivo, indicar cuantos dígitos tiene.***

***Solución*:** Vamos a realizar divisiones sucesivas entre 10 y al número se le va ir asignando su cociente. Ejemplo: Si tuviéramos el número num = 4512 hacemos:

cont ← *0*

num ← num div 10 , es decir
num ← *4512 div 10 lo cual quedaría*
num ← *451 ; num = 451* y seguidamente contabilizamos *cont* ← *cont+1; cont = 1*

num ← num div 10 , es decir
num ← *451 div 10 lo cual quedaría*
num ← *45 ; num = 45* y seguidamente contabilizamos *cont* ← *cont+1; cont = 1+1=2*

num ← num div 10 , es decir
num ← *45 div 10 lo cual quedaría*
num ← *4 ; num = 4* y seguidamente contabilizamos *cont* ← *cont+1; cont = 2+1=3*

num ← num div 10 , es decir
num ← *4 div 10 lo cual quedaría*
num ← *0 ; num = 0* y seguidamente contabilizamos *cont* ← *cont+1; cont = 3+1=4*

Aquí debemos parar, pues el contador ya contabilizó la cantidad de cifras que tenía el número inicial, por tanto, la condición debería ser: **Mientras** (num ≠ 0) **hacer**

Pseudocódigo Ejemplo
Var *num, cont:* entero
Inicio
Leer (*num*)
cont ← 0
Mientras (*num* ≠ 0) **hacer**
num ← *num* ***div*** 10
cont ← *cont+1*
fin_mientras
Escribir (*cont*)
Fin

Al ir asignando al número el cociente de su división entre 10, iremos disminuyendo la cantidad de cifras del número y a la vez aprovecharemos para contabilizar sus cifras.

Ejemplo: ***Diseñe un algoritmo para invertir un número entero (de cualquier cantidad de cifras). Una vez invertido, presentarlo.***

Solución: Ya se hizo el algoritmo para invertir un número, pero se debía conocer la cantidad de cifras que tenía el número. Ahora nos piden invertir un número, no importando la cantidad de cifras que tenga. Este ejercicio se resuelve con el uso de una estructura **mientras.**

Pseudocódigo Invertir
Var *inv, num, r* : entero
Inicio
Leer (*num*)
inv ← 0
Mientras (num ≠ 0) hacer
r ← *num* ***mod*** *10*
num ← *num* ***div*** *10*
inv ← *inv *10 + r*
fin_mientras
Escribir (*num, inv*)
Fin

No olviden inicializar con cero a la variable *inv*, que almacenará al número invertido.

IMPORTANTE: Fíjense que la variable *num*, va perdiendo su valor inicial (disminuye la cantidad de cifras) hasta ser igual a cero.

Ejemplo: ***Diseñe un algoritmo para invertir un conjunto de números enteros. (de cualquier cantidad de cifras). Una vez invertidos, presentarlos.*** **Solución:**

```
Pseudocódigo Invertir_varios_números
Var     i, inv, num, n, r : entero
Inicio
    Leer (n)
  desde i ← 1 hasta n
        Leer (num)
        inv ← 0
      Mientras (num ≠ 0) hacer
            r ← num mod 10
            num ← num div 10
            inv ← inv *10 + r
      fin_mientras
        Escribir (num, inv)
  fin_desde
Fin
```

Ejemplo: ***Diseñar un algoritmo para identificar si un número entero positivo es capicúa.***

Solución: Un numeral es capicúa si es igual a su forma invertida. Lo que haremos será invertir al número y compararlo con su forma inicial. Pero, recuerde que el valor de *num* se pierde y llega ser igual a cero. Debido a que necesitamos el valor de *num* intacto, utilizaremos una variable auxiliar, y va a ser a esta variable a la cual la vamos a invertir.

Pseudocódigo Capicúa
Var *aux, inv, num:* entero
Inicio
Leer (*num*)
aux ← *num*
inv ← 0
Mientras (*aux* ≠ 0) **hacer**
r ← *aux* ***mod*** *10*
aux ← *aux* ***div*** *10*
inv ← *inv *10 + r*
fin_mientras
Si (*inv = num*) **entonces**
Escribir ("*es capicúa*")
sino
Escribir ("*No es capicúa"*)
fin_si
Fin

Almacenamos el valor de *num* en *aux*, y empezamos a trabajar con aux, para que num mantenga su valor y no lo pierda.

Comparamos el número inicial con su forma invertida

Ejemplo: ***Diseñar un algoritmo para identificar de un conjunto de n números enteros positivos aquellos que son capicúas.***

Solución:

```
Pseudocódigo Capicúa
Var     n, i, aux, inv, num: entero
Inicio
    Leer (n)
  desde i ← 1 hasta n
      Leer (num)
      aux ← num
      inv ← 0
    Mientras (aux ≠ 0) hacer
        r ← aux mod 10
        aux ← aux div 10
        inv ← inv *10 + r
    fin_mientras
    Si (inv = num) entonces
        Escribir ("es capicúa")
    sino
        Escribir ("No es capicúa")
    fin_si
  fin_desde
Fin
```

Almacenamos el valor de *num* en *aux*, y empezamos a trabajar con aux, para que num mantenga su valor y no lo pierda.

Comparamos el número inicial con su forma invertida

Ejemplo: ***Diseñar un algoritmo que reciba n números enteros positivos, y que identifique a aquellos que son capicúas, mostrando un mensaje: "sí es capicúa". Al final, que muestre cuántos de los números ingresados fueron capicúas.***

Solución:

Pseudocódigo Capicúas
Var *n, i, aux, inv, num, cont:* entero
Inicio
Leer (*n*)
cont ← 0
desde i ← *1 hasta* n
Leer (*num*)
aux ← *num*
inv ← 0
Mientras (*aux* ≠ 0) **hacer**
r ← *aux* ***mod*** *10*
aux ← *aux* ***div*** *10*
inv ← *inv *10 + r*
fin_mientras
Si (*inv = num*) **entonces**
Escribir ("*es capicúa*")
cont ← *cont* +1
sino
Escribir ("*No es capicúa"*)
fin_si
fin_desde
Escribir ("*cantidad total de capicúas*: ", *cont*)
Fin

No olvidar inicializar el contador de capicúas

Almacenamos el valor de *num* en *aux*, y empezamos a trabajar con aux, para que num mantenga su valor y no lo pierda.

Comparamos el número inicial con su forma invertida

Ejemplo: ***Para un conjunto de n números enteros positivos, se desea saber ¿Cuántos de ellos son capicúas con el número de cifras impares?***

Solución:

```
Pseudocódigo Ejemplo
Var     i, n, r, cf, inv, num, aux: entero
Inicio
    Leer (n)
    cont ← 0
    desde i ← 1 hasta n
        cf ← 0
        Leer (num)
        inv ← 0
        aux ← num
        Mientras (aux ≠ 0) hacer
            r ← aux mod 10
            aux ← aux div 10
            inv ← inv *10 + r
            cf ← cf + 1
        fin_mientras
        Si (inv = num ∧ cf mod 2 ≠ 0) entonces
            cont ← cont + 1
        fin_si
    fin_desde
    Escribir (cont)
Fin
```

Ejemplo: ***Para un número entero positivo, determinar si es primo o no.***

Solución: Este método es del profesor Acosta. Recordar que N es primo si no es divisible entre 2, 3, … $\sqrt{N}$. Lo que se va a realizar es: Vamos a elevar al cuadrada a 2, 3,…, $\sqrt{N}$, para obtener 2^2, 3^2, … , N.

Pseudocódigo Ejemplo

```
Var     num, d: entero
        primo: lógico
Inicio
    Leer (num)
    d ← 2
    primo ← V
    Mientras (d*d < num ∧ primo) hacer
        Si(num mod d = 0) entonces
            primo ← F
        sino
            d ← d + 1
        fin_si
    fin_mientras
    Si (primo) entonces
        Escribir ("El número es primo")
    sino
        Escribir ("El número no es primo")
    fin_si
Fin
```

Ejemplo: ***A un conjunto de 100 alumnos se le asigna un código que oscila desde 500 hasta 599, luego de rendir un examen se obtiene la nota (0 a 20). Se desea averiguar la mayor nota, cuantas veces aparece y los códigos de los alumnos que lo obtuvieron.***

Solución:

```
Pseudocódigo Ejemplo
Var cod, nota, max, i, cont, lista: entero
Inicio
    desde i ← 1 hasta n
        Leer (cod, nota)
        Si (i = 1) entonces
            cont ← 1
            max ← nota
            lista ← cod
        sino
            Si (max < nota ) entonces
                max ← nota
                cont ← 1
                lista ← cod
            sino
                Si (nota = max) entonces
                    cont ← cont + 1
                    lista ← lista*1000  + cod
                fin_si
            fin_si
        fin_si
    fin_desde
     Escribir (max, cont)
    Mientras (lista ≠0) hacer
        cod ← lista mod 1000
        Escribir (cod)
        lista ← lista div 1000
    fin_mientras
Fin
```

Ejemplo: ***Diseñar un algoritmo que reciba n números enteros positivos , y que identifique a aquellos que son capicúas, mostrando un mensaje: "sí es capicúa". En el caso que no sea capicúa que muestre un mensaje: "No es capicúa" y que además aparezca la cantidad de cifras de este número, que muestre todos sus divisores y la cantidad de divisores de este número mostrando un mensaje si fue o no fue primo. Al final, que muestre cuántos de los números ingresados fueron capicúas, cuántos no fueron capicúas y cuantos fueron primos.***

Solución:

Pseudocódigo Ejemplo

Var *num, n, cont_cif, i, cont_div, cont_cap, cont_prim, inv, aux*: entero

Inicio

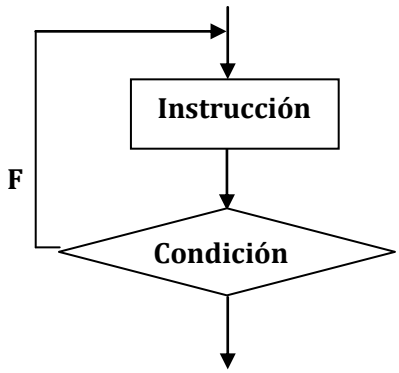
```
Repetir
    Leer (n)
Hasta_que (n > 0)
cont_prim ← 0
cont_cap ← 0
desde i ← 1 hasta n
    Repetir
        Leer (num)
    Hasta_que (num > 0)
    cont_cif ← 0
    inv ← 0
    aux ← num
    Repetir
        r ← aux mod 10
        aux ← aux div 10
        cont_cif ← cont_cif + 1
        inv ← inv * 10 + r
    Hasta_que (aux = 0)
    Si (num = inv) entonces
        Escribir ("Es capicúa")
        cont_cap ← cont_cap+1
```

```
        sino
            Escribir ("No es capicúa")
        fin_si
        Escribir ("Cantidad de cifras:  ", cont_cif)
        cont_div ← 0
        desde i ← 1 hasta num
            Si (num mod i = 0) entonces
                Escribir (i, " es divisor de ", num)
                cont_div ← cont_div + 1
            fin_si
        fin_desde
        Escribir ("Cantidad de divisores", cont_div)
        Si (cont = 2) entonces
            Escribir (num, " es primo")
            cont_prim ← cont_prim + 1
        Sino
            Escribir (num, " no es primo")
        fin_si
    fin_desde
    Escribir ("Cantidad de capicúas: ", cont_cap)
    Escribir ("Cantidad de no capicúas ", n – cont_cap)
    Escribir ("Cantidad de primos ", cant_prim)
Fin
```

8.3.3. BUCLE CON SALIDA CONTROLADA (ESTRUCTURA REPETITIVA REPETIR). ESTRUCTURA CON PRE-TEST DE PRUEBA

Como habrán notado, en las estructuras repetitivas **mientras**, si la condición es inicialmente falso, el cuerpo del bucle no se ejecutará ni una sola vez; pero existen muchas situaciones en las que se desea que un bucle se ejecute al menos una vez antes de comprobar la condición de repetición. Son en estas situaciones cuando se utiliza las estructuras repetitivas **repetir**, las cuales se ejecutarán sí o sí por lo menos una vez.

Funcionamiento: Con una estructura **repetir**, el cuerpo del bucle se ejecuta siempre al menos una vez. Cuando una instrucción **repetir** se ejecuta, lo primero que sucede es la ejecución del cuerpo del bucle, y a continuación, se evalúa la condición. Si se evalúa como falsa, el cuerpo del bucle se repite y la condición se evalúa otra vez. Solo cuando la condición resulte ser verdadera, allí el bucle termina.



Repetir
Instrucción
Hasta_que (*condición*)

OBSERVACIÓN

Debemos modificar la condición en el cuerpo del bucle para evitar que sea infinito.

USOS DE LA ESTRUCTURA REPETIR

a) Validación de la entrada de datos:

Validar que un dato sea positivo:

Repetir
Leer (*dato*)
Hasta_que (*dato* > 0)

Validar una nota de 0 a 20

Repetir
Leer (*nota*)
Hasta_que ($0 \leq nota \wedge nota \leq 20$)

Validar un código de tres cifras

Repetir
  Leer (*cod*)
Hasta_que (100 ≤ *cod* ∧ *cod* ≤ 999)

En cada uno de las estructuras anteriores, el programa se va a repetir indefinidamente y solo dejará de repetirse hasta que la condición resulte ser verdadera.

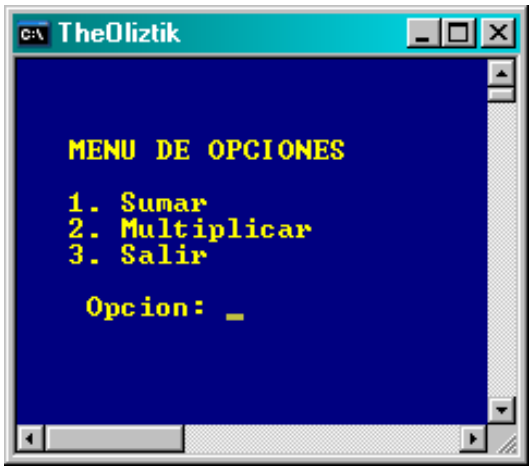
b) Para el uso de menús (opciones)

Ejemplo:

Mostrar una lista de opciones y terminar o salir si y solo sí se elija la opción correspondiente para tal acción.

Leer (*n1, n2*)
Repetir
  Escribir ("*Menú de Opciones*")
  Escribir (" *1. Sumar* ")
  Escribir (" *2. Multiplicar* ")
  Escribir (" *3. Salir* ")
  Repetir
    Escribir ("*opción: *")
    Leer (*opc*)
  Hasta_que (*opc* = 1 ∨ *opc* = 2 ∨ *opc* = 3)
  En_caso (*opc*)
    1: Escribir (*n1+n2*)
    2: Escribir (*n1 * n2*)
  fin_caso
Hasta_que (*opc = 3*)

Sea: n1 = 3 y n2 = 5



MÁXIMO COMÚN DIVISOR Y MÍNIMO COMÚN MÚLTIPLO

Ejemplo: ***Dados dos números enteros positivos, determine el máximo común divisor de dichos números.***

Solución:

Vamos a utilizar el algoritmo de Euclides para calcular el máximo común divisor (MCD) de dos números a y b.

Por ejemplo: Si los números fueran a = 24 y b = 18. Los pasos son los siguientes:

- Realizo la división entera de 24 entre 18.
- El divisor (18) pasa a convertirse en el dividendo.
- El resto de la división entera (6) pasa a convertirse en el divisor.

Este proceso se repite hasta que se obtiene resto = 0. Luego el MCD será el último divisor.

OBS: El algoritmo de euclides permite calcular el MCD de dos números sin importar quién es mayor o quién es menor. Funciona en cualquier orden.

Pseudocódigo MCD
Var *a, b, r*: entero
Inicio
  Leer (*a, b*)
  Repetir
    r ← *a* ***mod*** *b*
    a ← *b*
    b ← *r*
  Hasta_que (*r* = 0)
  Escribir (*a*)
Fin

La variable "a" almacena el mcd de los números.

Ejemplo: ***Dados nos números enteros positivos, determine el mínimo común múltiplo de dichos números.***

Solución: No vamos a calcular directamente el mínimo común múltiplo (MCM) de los números, sino que usaremos la siguiente propiedad. A partir del cálculo del MCD se podrá calcular el MCM.

MCD (a, b) x MCM (a, b) = a x b

Pseudocódigo MCM

Var *a, b, prod, mcm, r*: entero

Inicio

Leer (*a, b*)

prod ← *a* * *b*

Repetir

r ← *a* ***mod*** *b*

a ← *b*

b ← *r*

Hasta_que (*r* = 0)

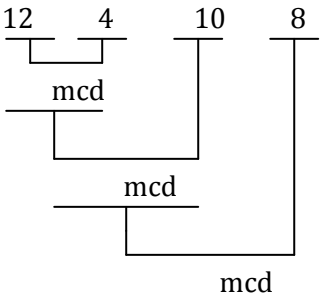
mcm ← *prod* ***div*** *a*

Escribir (*mcm*)

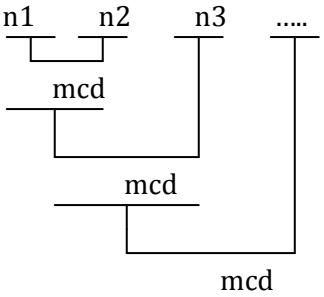
Fin

Ejemplo: ***Para n (n ≥2) números enteros positivos determine el MCD.***

Solución: Hallaremos el mcd tomando a los números de dos en dos, ejemplo:



El procedimiento general será:



Pseudocódigo MCD

Var *n, mcd, num, r, i:* entero

Inicio

Repetir

Leer (*n*)

Hasta_que (*n* ≥ 2)

desde i ← *1 hasta* n

Repetir

Leer (*num*)

Hasta_que (*num* > 0)

Si (*i* = 1) **entonces**

mcd ← *num*

sino

Repetir

r ← *mcd* ***mod*** *num*

mcd ← *num*

num ← *r*

Hasta_que (*r* = 0)

fin_si

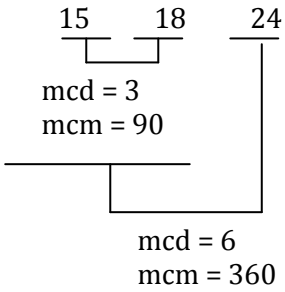
fin_desde

Escribir (*mcd*)

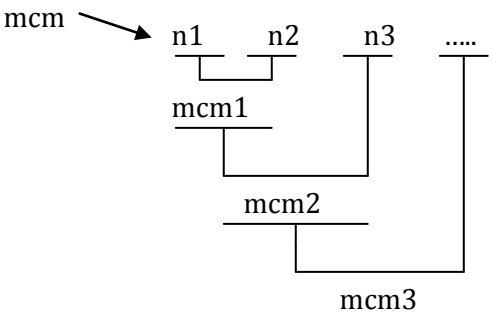
Fin

Ejemplo: ***Para n (n ≥2) números enteros positivos determine el MCM.***

Solución: Hallaremos el mcm tomando a los números de dos en dos, ejemplo:



El procedimiento general será:



Pseudocódigo MCM

Var *mcm, n, mcd, num, r, i:* entero

Inicio

- **Repetir**
  - **Leer** (*n*)
- **Hasta_que** (*n* ≥ 2)
- **desde** i ← *1 hasta* n
  - **Repetir**
    - **Leer** (*num*)
  - **Hasta_que** (*num* > 0)
  - **Si** (*i* = 1) **entonces**
    - *mcd* ← *num*
    - *mcm* ← *num*
  - **sino**
    - *mcm* ← *mcm* * *num*
    - **Repetir**
      - *r* ← *mcd* ***mod*** *num*
      - *mcd* ← *num*
      - *num* ← *r*
    - **Hasta_que** (*r* = 0)
    - *mcm* ← *mcm* ***div*** *mcd*
  - **fin_si**
- **fin_desde**
- **Escribir** (*mcm*)

Fin

Ejemplo: ***Determine la siguiente suma para x y n dados:***

$$S = \frac{x^2}{2!} - \frac{x^4}{4!} + \frac{x^6}{6!} - \ldots + (-1)^{n+1} \frac{x^{2n}}{(2n)!}$$

Pseudocódigo Suma

Var *x, S, term:* real

n, i: entero

Inicio

Leer (*x*)

Repetir

Leer (*n*)

Hasta_que (n>0)

S ← *0*

term ← -1

desde i ← *1 hasta* n

term ← *-1 * term * x * x / ((2 * i – 1) * (2 * i))*

S ← *S + term*

fin_desde

Escribir (*S*)

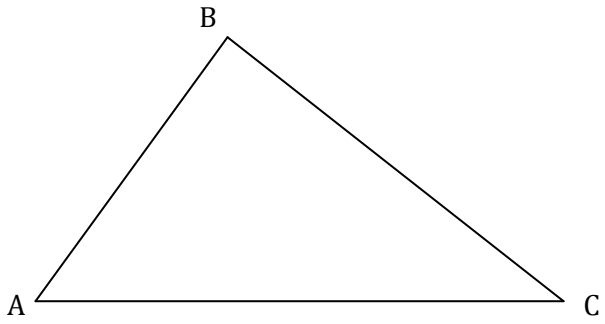
Fin

Ejemplo: **(Primera práctica califica, ciclo 97-I, 26-04-97)** ***Dado las ubicaciones (puntos no colineales en el plano) de tres clientes A, B, y C; se requiere determinar la ubicación de un centro comercial que equidiste de estas tres ciudades.***

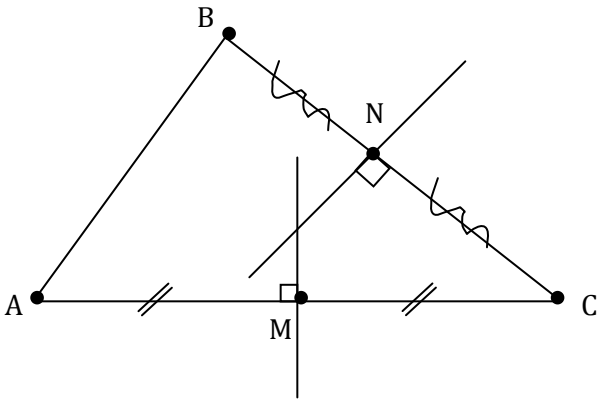
Solución:

ANÁLISIS DEL PROBLEMA: Este problema requiere conocimientos de Geometría Analítica.

La ubicación del centro comercial viene a ser el circuncentro del triángulo de vértices A, B, y C. Para determinar las coordenadas del circuncentro vamos a recurrir a la geometría analítica. Unimos mediante 3 segmentos a los 3 puntos no colineales (A, B y C) formando de esta manera un triángulo. (de cualquier tipo: rectángulo u oblicuángulo.



Trazamos las dos mediatrices relativas a los lados AC y BC.



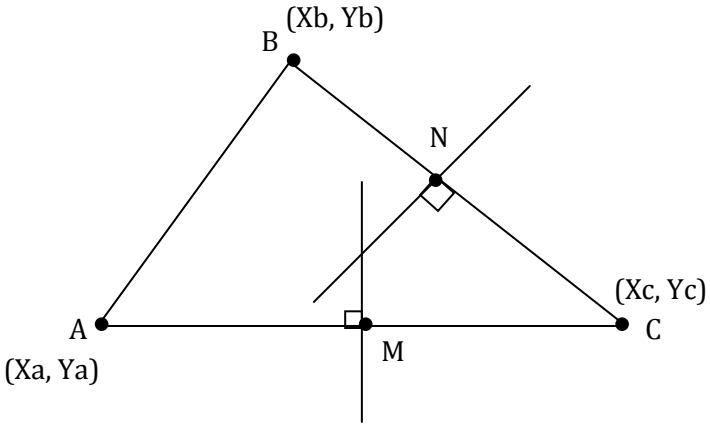
El punto de intersección de estas dos rectas mediatrices vendría a ser el Circuncentro del triángulo, y es además el punto que equidita de los tres vértices.

Sean las coordenadas de los vértices:

A = (Xa, Ya)

B = (Xb, Yc)

C = (Xc, Yc)



Hallamos las coordenadas de los puntos M y N:

$$M = \frac{A+C}{2} = \left(\frac{Xa+Xc}{2}; \frac{Ya+Yc}{2}\right)$$

$$N = \frac{B+C}{2} = \left(\frac{Xb+Xc}{2}; \frac{Yb+Yc}{2}\right)$$

Además podemos conocer los vectores $\overrightarrow{BC}$ y $\overrightarrow{AC}$ y sus ortogonales

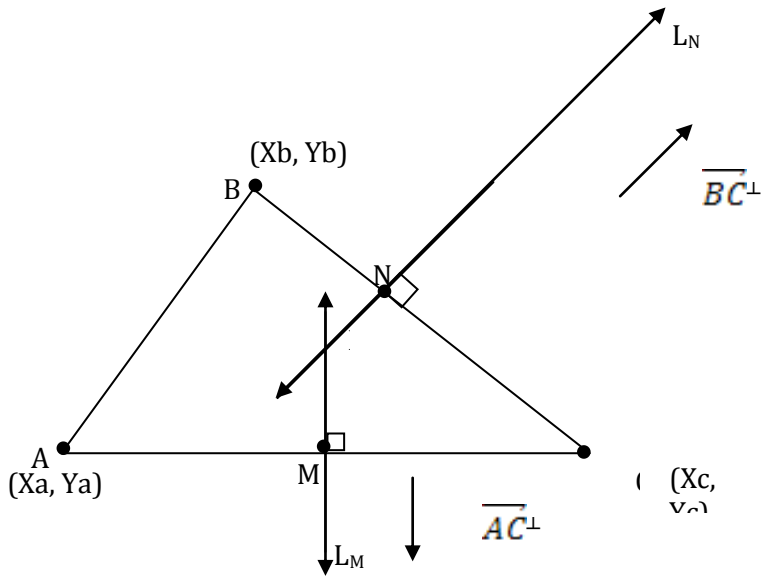
$$\overrightarrow{BC} = C - B = (Xc - Xb; Yc - Yb) \Rightarrow \qquad \overrightarrow{BC}^{\perp} = (Yb - Yc; Xc - Xb)$$

$$\overrightarrow{AC} = C - A = (Xc - Xa; Yc - Ya) \Rightarrow \qquad \overrightarrow{AC}^{\perp} = (Ya - Yc; Xc - Xa)$$

Sean L_N y L_M dos rectas cuyas ecuaciones vectoriales son las siguientes:

L_N: P = N + t $\overrightarrow{BC}^{\perp}$ (recta que pasa por N y con vector direccional $\overrightarrow{BC}^{\perp}$)

L_M: P = M + r $\overrightarrow{AC}^{\perp}$ (recta que pasa por M y con vector direccional $\overrightarrow{AC}^{\perp}$)



Al ser P un punto en común de ambas rectas, entonces P debe satisfacer ambas ecuaciones vectoriales, por eso igualamos las ecuaciones y calculamos

$$N + t\ \overrightarrow{BC}^{\perp} = M + r\ \overrightarrow{AC}^{\perp}$$

$$t\ \overrightarrow{BC}^{\perp} = \overrightarrow{NM} + r\ \overrightarrow{AC}^{\perp}$$

$$t = \left(\frac{\overrightarrow{NM}\ \overrightarrow{AC}}{\overrightarrow{BC}^{\perp}.\ \overrightarrow{AC}}\right.$$

$$P = N + \frac{\overrightarrow{NM}\ \overrightarrow{AC}}{\overrightarrow{BC}^{\perp}.\ \overrightarrow{AC}})\ \overrightarrow{BC}^{\perp}$$

$$P = \left(\frac{Xb+Xc}{2};\frac{Yb+Yc}{2}\right) + \left(\frac{\left(\frac{Xa-Xb}{2};\frac{Ya-Yb}{2}\right).\ (Xc-Xa;Yc-Ya)}{(Yb-Yc;Xc-Xb).(Xc-Xa;Yc-Ya)}\right)\ (Yb-Yc;Xc-Xb)$$

Ya tenemos calculado P en función de las coordenadas de los puntos A, B y C.

Ejemplo: ***Se desde leer un conjunto de n (n ≥ 30) códigos enteros (cada código ≥ 10000). Se desea averiguar cuántos de dichos códigos tiene la secuencia 2, 3, 7 no necesariamente adyacentes.***

Solución: Ejemplo: 213746 (sí), 327541 (no), 2372374 (sí)

Pseudocódigo Ejemplo

Var *cont, num, i, n, r, d, j:* entero

sec: lógica

Inicio

Repetir

Leer (*n*)

Hasta_que (*n ≥ 30*)

cont ← *0*

desde i ← *1 hasta* n

Repetir

Leer (*cod*)

Hasta_que (*cod ≥ 10000)*

j ← *1*

sec ← V

Mientras (j ≤ 3 ∧ *sec*) **hacer**

En_caso (*j*)

1: *d* ← *7*

2: *d* ← *3*

3: *d* ← *2*

fin_caso

Repetir

r ← *cod* ***mod*** *10*

cod ← *cod* ***div*** *10*

Hasta_que (*cod = 0* ∨ *r = d*)

Si (*cod = 0*) **entonces**

sec ← F

fin_si

j ← *j + 1*

fin_mientras

```
        Si (sec) entonces
            cont ← cont + 1
        fin_si
    fin_desde
    Escribir (cont)
Fin
```

Ejemplo: ***Algoritmo para pasar un numeral de base 2 a base 10***. **Solución:**

```
Pseudocódigo Binario_Decimal
Var     C, d, S, num: entero
Inicio
    Repetir
        Leer (num )
    Hasta_que (num > 0)
    S ← 0
    C ← 1
    Repetir
        d ← num mod 10
        num ← num div 10
        S ← d * C + S
        C ← C * 2
    Hasta_que (num = 0)
    Escribir ( S)
Fin
```

Ejemplo: ***Algoritmo para transformar un numeral de una base n menor que 10 a la base 10.***

Solución:

```
Pseudocódigo Numeración
Var     C, d, S, num, n: entero
Inicio
   Repetir
      Leer (n, num)              (* Base del numeral *)
   Hasta_que (2 ≤ n ^ n ≤ 9 ^ num > 0)
      S ← 0
      C ← 1
   Repetir
      d ← num mod 10
      num ← num div 10
      S ← d * C + S
      C ← C * n
   Hasta_que (num = 0)
    Escribir (S)
Fin
```

Ejemplo: ***Algoritmo para transformar un numeral de base 10 a una base n menor que 10.***

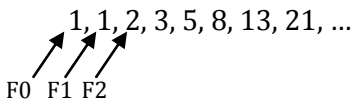
Solución:

```
Pseudocódigo Cambio
Var     num, n, N, P, r: entero
Inicio
   Repetir
      Leer (num)
   Hasta_que (num > 0)
   Repetir
      Leer (n)                    (* Base del numeral *)
   Hasta_que (2 ≤ n ∧ n ≤ 9)
      N ← 0
      P ← 1
   Repetir
      r ← num mod n
      num ← num div n
      N ← N + r * P
      P ← P * 10
   Hasta_que (num < n)
     N ← N + num * P
     Escribir (N)
Fin
```

Ejemplo: ***Dado un número natural positivo, indique si es un número perfecto. (NOTA: un número es perfecto si la suma de sus divisores (sin incluir al mismo número) resulta igual a sí mismo. Ejemplo: 6 es perfecto pues 6 = 1+2+3.*** **Solución:**

```
Pseudocódigo Perfecto
Var     n, S, i: entero
Inicio
    Repetir
        Leer (n)
    Hasta_que (n > 0)
    S ← 0
    Desde i ← 1 hasta n-1
        Si (n mod i = 0) Entonces
            S ← S + i
        fin_si
    fin_desde
    Si (S = n) entonces
        Escribir ("Sí es perfecto")
    Sino
        Escribir ("No es perfecto")
    fin_si
Fin
```

Ejemplo: ***Diseñe un algoritmo que entregue el término enésimo de la serie de Fibonacci. (Tomando a n a partir de cero.)***



Solución: F_0 y F_1 van a ser fijos: $F_0 = 1$; $F_1 = 1$; $F_n = F_{n-1} + F_{n-2}$ para $n \geq 2$

```
Pseudocódigo Fibonacci
Var     F, F0, F1: entero
Inicio
    F0 ← 1
    F1 ← 1
    Repetir
        Leer (n)
    Hasta_que (n ≥ 0)
    Si (n = 0 ∨ n = 1) entonces
        F ← 1
    Sino
        Desde i ← 2 hasta n
            F ← F1+F0
            F0 ← F1
            F1 ← F
        fin_desde
    fin_si
    Escribir (F)
Fin
```

Ejemplo: ***Dado un conjunto de n alumnos (hombres y mujeres) se desea leer el código (entero de 3 dígitos), nota (de 0 a 20) y sexo (M o F). Luego averiguar:***

a) El alumno (varón) con la mayor nota y menor código (respecto a esa mayor nota). El número de alumnos que tienen esa mayor nota.
b) La alumna con la menor nota y mayor código (respecto a esa menor nota). El número de alumnas que tienen la misma menor nota.

Pseudocódigo Ejercicio
Var *n, i, cod, nota, maxnotH, minnotM, mincodH, maxcodM, contH, contM*: entero
sexo: caracter
Inicio
Repetir
Leer (*n*)
Hasta_que (*n* > 0)
maxnotH ← 0
minnotaM ← 20
mincodH ← 999
maxcodM ← 100
contH ← 0
contM ← *0*
desde i ← 1 *hasta* n
Repetir
Leer (*cod*)
Hasta_que (100 ≤ *cod* ∧ *cod* ≤ 999)
Repetir
Leer (*nota*)
Hasta_que (0 ≤ *nota* ∧ *nota* ≤ 20)
Leer (*sexo*)
Si (*sexo* = 'M') **entonces**
Si (*maxnotH* < *nota*) **entonces**
maxnotH ← *nota*
contH ← 1
mincodH ← cod
sino
Si (*maxnotH* = *nota*) **entonces**
contH ← *contH* + 1
Si (*mincodH* > *cod*) **entonces**
mincodH ← *cod*

```
                    fin_si
                fin_si
            fin_si
        sino
            Si (sexo = 'F') entonces
                Si (minnotM > nota) entonces
                    minnotM ← nota
                    contM ← 1
                    maxcodM ← cod
                sino
                    Si (minnotM = nota) entonces
                        contM ← contM + 1
                        Si (maxcodM < cod) entonces
                            maxcodM ← cod
                        fin_si
                    fin_si
                fin_si
            fin_si
        fin_si
    fin_desde
    Escribir (maxnotH, mincodH, contH)
    Escribir (minnotM, maxcodM, contM)
Fin
```

Ejemplo: (2006-I)

Se tiene el código (entero de 6 cifras) de un grupo de n empleados entre varones y mujeres. Diseñe un algoritmo que reciba dichos códigos y luego determine:

a. El número de varones cuyo código termina en 41

b. El número de mujeres cuyos tres últimos dígitos de su código sea un número capicúa (número capicúa es aquel que se lee igual de derecha a izquierda que de izquierda a derecha)

Solución:

```
Pseudocódigo Ejemplo
Var         cod, cont_M, cont_F, inv, aux, r: entero
            sexo: caracter
Inicio
  Repetir
     Leer (n)
  Hasta_que (n>0)
   cont_M <- 0
   cont_F <- 0
  Desde i<-1 hasta n
     Leer (sexo)
    Repetir
       Leer (cod)
    Hasta_que (100000 <= cod y cod <= 999999)
    Si (sexo = 'M') Entonces
      Si (cod mod 100 = 41) Entonces
         cont_M <- cont_M + 1
      fin_si
    Sino (* sexo = 'F' *)
       inv <- 0
       aux <- cod mod 1000
      Mientras (aux <> 0) hacer
         r <- aux mod 10
         aux <- aux div 10
         inv <- inv * 10 + r
      fin_mientras
      Si (inv = cod) Entonces
         cont_F <- cont_F + 1
      fin_si
    fin_si
  fin_desde
    Escribir (cont_F, cont_M)
Fin
```

Ejemplo: ***Se tiene las notas de la 1era PC del curso de algoritmos de un grupo de alumnos que pertenecen a diferentes secciones (U, V, W). Diseñar un algoritmo para ingresar la nota y sección de dichos alumnos (el ingreso de datos termina cuando se ingresa como nota el número -10) y luego determine:***

a. El porcentaje de aprobados por sección (Si existen alumnos en dicha sección).
b. La nota promedio general (de todas las secciones)
c. La mayor nota y el número de alumnos que la poseen (de todas las secciones).

Solución:

Pseudocódigo Prob

```
Var       nota, cont_U, cont_V, cont_W, cont, S, max, cont_max : entero
          i, cont_ap_U, cont_ap_V, cont_ap_W : entero
          sección : carácter
Inicio
  cont ← 0
  cont_U ← 0 : cont_V ← 0 : cont_W ← 0
  cont_max ← 0
  S ← 0 : i ← 0
  cont_ap_U ← 0 : cont_ap_V ← 0 : cont_ap_W ← 0
  Repetir
    Repetir
      Leer (nota)
    Hasta_que (0 <= nota ∧ nota <= 20 ∨ nota = -10)
    Si (nota <> -10) Entonces
      i ← i + 1
      Repetir
        Leer (sección)
      Hasta_que ('U' <= sección ∧ sección <= 'W')
      S ← S + nota (* Suma todas las notas*)
      cont ← cont + 1 (* Cuenta la cantidad total de alumnos*)
      Si (i = 1) Entonces
        max ← nota
      Sino (* i = 2, 3, 4, ... n *)
        Si (max < nota) Entonces
          max ← nota
          cont_max ← 1
        Sino
          Si (max = nota) Entonces
            cont_max ← cont_max + 1
          fin_si
        fin_si
      fin_si
      En_caso (sección)
          'U': cont_U ← cont_U + 1
              Si (nota >= 10) Entonces
                  cont_ap_U ← cont_ap_U + 1
              fin_si
```

```
        'V': cont_V ← cont_V + 1
            Si (nota >= 10) Entonces
                cont_ap_V ← cont_ap_V + 1
            fin_si
        'W': cont_W ← cont_W + 1
            Si (nota >= 10) Entonces
                cont_ap_W ← cont_ap_W + 1
            fin_si
    fin_caso
  fin_si
Hasta_que (nota >= -10)
Si (cont_U <> 0) Entonces
    Escribir("Aprobados sección U: ", ((cont_ap_U / cont_U)*100), "%")
fin_si
Si (cont_V <> 0) Entonces
    Escribir("Aprobados sección V: ", ((cont_ap_V / cont_V)*100), "%")
fin_si
Si (cont_W <> 0) Entonces
    Escribir("Aprobados sección W: ", ((cont_ap_W / cont_W)*100), "%")
fin_si
Si (cont <> 0) Entonces
    Escribir ("Nota promedio general: ", S/cont)
fin_si
Si (cont_max <> 0) Entonces
    Escribir ("Máxima nota: ", max, " lo poseen: ", cont_max)
fin_si
Fin
```

Ejemplo: (2007- II)

Diseñe un algoritmo que permita leer hasta n edades en un rango de 10 a 99 (la lectura de las edades termina cuando se ingresa la enésima edad o cuando se ingresa una edad fuera del rango); luego muestre:

a). La mayor y la menor edad, indicando el número de apariciones de las mismas. Indique también la secuencia de estas edades (primero las menores y luego las mayores)

b). Todas las edades ingresadas sin incluir la mayor y la menor.

Por ejemplo: para n = 13, las edades a ingresar son:

20, 12, 40 ,60, 55, 60, 35, 12, 30, 60, 8, 13, 105

Como la edad 8 está fuera del rango, sólo se leen las 10 primeras. Luego:
La salida de la parte a) es:

Menor	*:12*	*Apariciones: 2*
Mayor	*: 60*	*Apariciones: 3*
Secuencia	*: 1212606060*	

La salida de la parte b) es:

30, 35, 55, 40, 20

Pseudocódigo Problema

```
Var     n, max , cont_max , min, cont_min , sec1 , sec2 , i, edad, sec2, sec1, r: entero
        sigue: lógico
Inicio
  Repetir
     Leer (n)
  Hasta_que (n > 0)
   max ← 0  :  cont_max ← 0
   min ← 0  :  cont_min ← 0
   sec1 ← 0                    (* secuencia de la mayor y menor nota)
   sec2 ← 0                    (* secuencia de todos*)
   i ← 1  :  sigue ← V
  Mientras (i <= n ∧ sigue = V) Hacer
     Leer (edad)
    Si (edad < 10 ∨ edad > 99) Entonces
       sigue ← F
    Sino   (* Datos incorrectos *)
      Si (i = 1) Entonces
         max ← edad  :  cont_ max ← 1
         min ← edad  :  cont_min ← 1
      Sino
        Si (max < edad) Entonces
           max ← edad
           cont_max ← 1
        Sino
          Si (max = edad) Entonces
             cont_max ← cont_max + 1
          fin_si
        fin_si
        Si (min > edad) Entonces
           min ← edad
           cont_min ← 1
        Sino
          Si (min = edad) Entonces
             cont_min ← cont_min + 1
          fin_si
        fin_si
      fin_si
      sec2 ← sec2 * 100 + edad
    fin_si
    i ← i + 1
  fin_mientras
  Si (min <> 0) Entonces
     Escribir ("Menor: ", min, "Apariciones: ", cont_min)
     Desde i ← 1 hasta cont_min
        sec1 ← sec1 * 100 + min
```

```
    fin_desde
  fin_si
  Si (max <> 0) Entonces
    Escribir ("Mayor: ", max, "Apariciones: ", cont_max)
    Desde i ←1 hasta cont_max
      sec1 < sec1 * 100 + max
    fin_desde
  fin_si
  Si (sec1 <> 0) Entonces
    Escribir (sec1)
  fin_si
  Mientras (sec2 <> 0) Hacer
    r ← sec2 mod 100
    sec2 ← sec2 div 100
    Si (r <> max ∧ r <> min) Entonces
      Escribir (r)
    fin_si
  fin_mientras
Fin
```

Ejemplo: ***Diseñe un algoritmo que permita leer los códigos de un conjunto de n alumnos. Cada código está formado por 8 dígitos y contiene la siguiente información:***

- El primer dígito representa el sexo (Si es PAR será masculino y si es IMPAR será femenino).

- Los cuatro siguientes dígitos representan el año de nacimiento. Este año debe encontrarse en el rango de 1980 a 2007; en caso de no cumplirse, se volverá a leer el código.

- Los últimos 3 dígitos representan el número total de créditos llevados hasta el 2007.
Se pide que:

a) Determine el número de alumnos de sexo masculino que tengan la mayor edad.

b) Determine el número de alumnos de sexo femenino que tengan menor de 90 créditos.

Solución:

```
Pseudocódigo Problema

Var     n, i,  cod, Año, cont_M, cont_F: entero

Inicio
  Repetir
     Leer (n)
  Hasta_que (n > 0)
  Desde i ← 1 hasta n
     Repetir
        Repetir
           Leer (cod)
        Hasta_que (10000000 <= cod ∧ cod <= 99999999)
     Año ← (cod div 1000) mod 10000
     Hasta_que (1980 <= Año ∧ Año <= 2007)
     sexo ← cod div 10000000
     Si (sexo mod 2 = 0) Entonces              (* par =  masculino *)
        Si ((2007 - Año) >= 18) Entonces
           cont_M ←cont_M + 1
        fin_si
     Sino                                      (* impar = femenino *)
        Si (cod mod 1000 < 90) Entonces
           cont_F ← cont_F + 1
        fin_si
     fin_si
  fin_desde
    Escribir (cont_M, cont_F)
Fin
```

Ejemplo: ***Para un conjunto de n alumnos se desea leer por cada uno: Código (entero de 3 cifras) y nota. Luego presentar la menor nota indicando el código del alumno que lo posee.***

Por ejemplo: Para n = 4, se ingresan los siguientes datos:

Código	Nota
101	12
102	05
130	15
120	05

Luego la salida es:

Código	Nota
102	05
120	05

Solución:

```
Pseudocódigo Problema
Var          i, n, cod, nota, min, sec : entero
Inicio
  Repetir
     Leer (n)
  Hasta_que (n > 0)
  Desde i ← 1 hasta n
     Repetir
        Leer (cod)
     Hasta_que (100 <= cod ∧ cod <= 999)
     Repetir
        Leer (nota)
     Hasta_que (0 <= nota ∧ nota <= 20)
     Si (i = 1) Entonces
        min ← nota
        sec ← sec * 100 + cod          (*  sec ← cod *)
     Sino                              (*  i = 1, 2 , 3 ..  *)
        Si ( min > nota) Entonces
           min ← nota
           sec ← cod
        Sino
           Si (min = nota) Entonces
              sec ← sec * 100 + cod
           fin_si
        fin_si
     fin_si
  fin_desde
  Mientras (sec <> 0) hacer
     Escribir (sec mod 100)
     Escribir (min)
     sec ← sec div 100
  fin_mientras
Fin
```

RESOLUCIÓN DE PRIMERAS PRÁCTICAS CALIFICADAS

PRIMERA PRÁCTICA CALIFICADA - UNI CICLO 2010 II

Lima 13 de setiembre del 2010.

PROBLEMA 1

Diseñe un algoritmo que permita recibir 5 números enteros diferentes y luego nos presente el tercero mayor. Ejemplo:

Entrada: 5, 13, 10, 6, 7

Salida: el tercero mayor es 7 **Solución:**

Pseudocódigo Tercero_Mayor

Var *a, b, c, d, e, aux:* entero

Inicio

Leer (*a, b, c, d, e*)

Si (*a* < *e*) **entonces**

aux ← *a*
a ← *e*
e ← *aux*

fin_si

Si (*b* < *d*) **entonces**

aux ← *b*
b ← *d*
d ← *aux*

fin_si

Si (*a* < *b*) **entonces**

aux ← *a*
a ← *b*
b ← *aux*

fin_si

Si (d < *e*) **entonces**

aux ← *d*
d ← *e*
e ← *aux*

fin_si

Si (*b* < *d*) **entonces**

aux ← *b*
b ← *d*
d ← *aux*

fin_si

```
Si ( c > a) entonces
    Escribir (b, "es el tercero mayor")
sino                    ( * c ≤ a  * )
    Si ( c > b) entonces
        Escribir ( b, "es el tercero mayor")
    sino                ( * c ≤ b  * )
        Si (c > d) entonces
            Escribir (c, "es el tercero mayor")
        sino            ( * c ≤ d  * )
            Si (c > e ∨ c < e) entonces
                Escribir (d, "es el tercero mayor")
            fin_si
        fin_si
    fin_si
fin_si
Fin
```

PROBLEMA 2

Diseñe un algoritmo que reciba una cantidad no determinada de números enteros positivos de 4 a más cifras (el ingreso de datos termina cuando se ingresa el número -1), luego muestre:

a) La cantidad de números capicúas ingresados (NOTA: un número es capicúa si se lee igual de izquierda a derecha como de derecha a izquierda. Ejemplo: 131 es capicúa)

b) Los números primos ingresados y el promedio de los mismos.

Pseudocódigo Problema_2

Var *num, aux, NI, r, cont_cap, i, r1, d, cont_p, suma_p*: entero

prom_o: real

Inicio

cont_cap ← *0*

cont_p ← *0*

suma_p ← *0*

Repetir

Repetir

Leer (*num*)

Hasta_que (*num* ≥ 1000 ∨ *num* = -1)

Si (*num* ≠ *-1*) **entonces**

aux ← *num*

NI ← 0

Mientras (*aux* ≠ *0*) **hacer**

r ← *aux* ***mod*** *10*

aux ← *aux* ***div*** *10*

NI ← *NI* * 10 + *r*

fin_mientras

Si (*NI = num*) **entonces**

cont_cap ← *cont_cap* + 1

fin_si

d ← 0

r1 ← 0

```
        desde i ← 1 hasta num
            r1 ← num mod i
            Si (r1 = 0) entonces
                d ← d + 1
            fin_si
        fin_desde
        Si (d = 2) entonces
            Escribir (num, "es primo")
            cont_p ← cont_p + 1
            suma_p ← suma_p + num
        fin_si
    fin_si
Hasta_que (num = -1)
  Escribir ("La cantidad de capicúas ingresados fue: ", cont_cap)
  Si (cont_p = 0) entonces
      Escribir ("no se ingresó ningún primo, el promedio es cero")
  sino
      prom_p ← suma_p / cont_p
      Escribir ("el promedio es: ", prom_p, "de : ", cont_p, "primos ingresados")
  fin_si
Fin
```

PROBLEMA 3

En el mundial Sudáfrica 2010 participaron 32 equipos por diferentes países. Cada equipo tiene un cuadro de 40 jugadores. Por cada jugador se tienen los siguientes datos: código (entero de 4 cifras), edad y el número de goles anotados en este mundial.

Se pide diseñar un algoritmo que lea los datos de los jugadores y muestre:

a) Edad promedio de los jugadores por cada equipo y la edad promedio de todos los jugadores.

b) El número de jugadores con la mayor cantidad de goles anotados en el mundial.

Solución:

Datos de entrada: Número de equipos (32). Número de jugadores (40). Por cada jugador: código (de 4 cifras), edad, número de goles anotados.

Datos de salida: Edad promedio por equipo, edad promedio de todos los jugadores; número de jugadores con la mayor cantidad de goles anotados en el mundial.

Proceso:

- Leer el código, edad, número de goles
- Sumar las edades de los jugadores en un equipo y dividir el resultado entre 40, luego mostrar dicho resultado que sería el promedio.
- Sumar las sumas de edades de los jugadores de cada equipo, así tendría la suma de todos, luego muestro el resultado de haberlo dividido entre 32x40
- Contabilizar la cantidad de jugadores con el mayor número de goles en el mundial, primero en cada equipo y luego entre todos los equipos.

Pseudocódigo Mundial

Var *edad, código, num_gol, suma_edad, max_num_gol, cont1, edad_pe, i, j, suma_edad_total, edad_pt, cont2, max_num_gol_mund*: entero

Inicio

```
suma_edad_total ← 0
cont2 ← 0
desde i ← 1 hasta 32
    cont1 ← 0
    suma_edad ← 0
    desde i ← 1 hasta 40
        Repetir
            Leer (edad, código, num_gol)
        Hasta_que (edad > 0 ∧ edad < 100 ∧ num_gol ≥ 0 ∧ código ≥1000 ∧ código ≤ 9999)
        sumad_edad ← suma_edad + edad
        Si (j = 1) entonces
            max_num_gol ← num_gol
            cont1 ← 1
        sino
            Si (max_num_gol < num_gol) entonces
                max_num_gol ← num_gol
                cont1 ← 1
            sino
                Si (max_num_gol = num_gol) entonces
                    cont1 ← cont1 + 1
                fin_si
            fin_si
        fin_si
    fin_desde
    edad_pe ← suma_edad mod 40
    Escribir ("la edad promedio del equipo número ", i, " es ", edad_pe)
    suma_edad_total ← suma_edad_total + suma_edad
    Si (i = 1) entonces
        max_num_gol_mund ← max_num_gol
        cont2 ← cont1
    sino
        Si (max_num_gol_mund < max_num_gol) entonces
            max_num_gol_mund ← max_num_gol
            cont2 ← cont1
```

```
        sino
            si (max_num_gol_mund = max_num_gol) entonces
                max_num_gol_mund ← max_num_gol
                cont2 ← cont2 + 1
            fin_si
        fin_si
    fin_si
fin_desde
edad_pt ← suma_edad_total div 1280
Escribir ("el promedio de las edades de los 1280 jugadores es: ", edad_pt)
Escribir ("el número de jugadores con la mayor cantidad de goles anotados en el mundial es: ",
    cont2, " y el número máximo de goles fue", max_num_gol_mund)
```

Fin

PRIMERA PRÁCTICA CALIFICADA - UNI CICLO 2010 I

Lima 26 de abril del 2010.

PROBLEMA 1

Diseñe un algoritmo que permita leer un número entero positivo N mayor o igual a 10 y luego muestre:

a) La secuencia (número M) formada por los divisores impares de 1 cifra.

b) La secuencia (número P) formada por los divisores pares de 2 cifras, verificando si P es perfecto.

Ejemplo: *Entrada: N=40*

Salidas: M=51

P=402010. P no es perfecto

Solución:

Pseudocódigo Problema1

```
Var   r, N, i, aux, inv1, inv2, sec1, sec2, suma_div, r1, r2, cf : entero
Inicio
  Repetir
    Leer (N)                    (*Asegurando que n sea mayor o igual que 10*)
  Hasta_que(N>=10)
  sec1 ← 0
  sec2 ← 0
  suma_div ← 0                  (* Desde el 1 hasta el mismo N vamos a dividirlo y en
                                   aquellos casos en los que el residuo es cero, entonces
                                   i será divisor de N *)
  desde i ← 1 hasta N
    aux ← i
    cf ← 0
    r ← N mod i
    Si (r = 0) entonces
      Mientras (aux ≠ 0) hacer  (* En este bucle contamos las cifras del divisor i
        aux ← aux div 10
        cf ← cf+1
      fin_mientras
      Si (cf=1 ∧ i mod 2 ≠ 0) entonces   (* Si i es un divisor con una cifra y a la
        sec1 ← sec1 *10+i                   vez es impar, entonces formamos un
      fin_si                                numeral con este, es decir, una
      Si (cf=1 ∧ i mod 2 ≠ 0) entonces   secuencia1 *)
        sec1 ← sec1 *10+i
      fin_si
```

```
        Si (cf=2 ∧ i mod 2 = 0) entonces
            sec2 ← sec2 *100+i
        fin_si
        Si (i < N) entonces
            suma_div ← suma_div + i
        fin_si
    fin_si
fin_desde
inv1 ← 0
Repetir
    r1 ← sec1 mod 10
    sec1 ← sec1 div 10
    inv1 ← inv1*10 + r1
Hasta_que (sec1 = 0)
Escribir (inv1)
inv2 ← 0
Repetir
    r2 ← sec2 mod 10
    sec2 ← sec2 div 10
    inv2 ← inv2*10 + r2
Hasta_que (sec2 = 0)
Escribir (inv2)
Si (suma_div = N) entonces
    Escribir ("Perfecto")
Sino
    Escribir ("No es perfecto")
fin_si
Fin
```

(* Si i es un divisor con dos cifras y a la vez es par, entonces formamos un numeral con este, es decir, una secuencia2 *)

(* Sumamos los divisores i menores que N y los vamos acumulando *)

(* En este bucle invertimos la secuencia1 formada con aquellos divisores con una cifra e impar *)

(* En este bucle invertimos la secuencia2 formada con aquellos divisores con dos cifras y pares *)

PROBLEMA 2

Diseñe un algoritmo que permita ingresar una cantidad no determinada de números enteros positivos mayores o iguales a 10 (el ingreso termina cuando se ingresa el número 0), luego muestre:

a) El menor número primo ingresado y el número de veces que éste aparece.

b) El número de veces que en forma consecutiva aparecen los números 25, 12 y 2010.

Solución:

```
Pseudocódigo Problema2
Var     n, cont, mín_primo, cont_sec, a, b, c, i : entero
        primero : lógico
Inicio
    primero ← V
    mín_prim ← 0
    cont_sec ←0
    a ←0
    b ←0
    c ←0
  Repetir
    Repetir
        Leer(n)
    Hasta_que (n >= 10  v  n = 0)
    Si (n ≥ 10) entonces
        cont ← 0
      desde i ← 1 hasta n
        Si (n mod i = 0) entonces
            cont ← cont + 1
        fin_si
      fin_desde
      Si (cont = 2) entonces
        Si (primero) entonces
            mín_primo ← n
            primero ← F
        Sino
            Si (mín_primo > n) entonces
                mín_primo ←n
            fin_si
        fin_si
      fin_si
      Si (n = 25) entonces
        a ← 25
      Sino
        Si (a = 25 ∧ n = 12) entonces
            b ← 12
```

Sino
Si ($a = 25 \wedge b = 12 \wedge n = 2010$) **entonces**
cont_sec ← *cont_sec* + 1
fin_si
fin_si
fin_si
fin_si
Hasta_que (n = 0)
Si (*mín_primo* > 0) **entonces**
Escribir ("*El menor número primo ingresado fue*: ", *mín_primo*)
Sino
Escribir ("*No hay primos*")
fin_si
Escribir ("El *# de veces que en forma consecutiva aparecen 25, 12 y 2010* es: ", *cont_sec*)
Fin

PROBLEMA 3

Diseñe un algoritmo que reciba 4 números enteros positivos menores que 16 y luego retorne los números modificados. La modificación se realiza utilizando el sistema binario (ver ejemplo).

Ejemplo:

Entrada: Números: 11, 6, 11, 7

Número decimal		Número binario
11	=	1011
6	=	0110
11	=	1011

Número decimal		Número binario
1010	=	10
0101	=	5
1111	=	15

Salida: Números: 10, 5, 15, 11

Solución: *Hay que tener presente que los números ingresados inicialmente deben de ser menores que 16, por lo tanto su representación en sistema binario tendrá como máximo 4 dígitos.*

```
Pseudocódigo Problema3
Var     a, b, c, d, i, num, pot, binar, r, N1, N2, N3, N4, decimal, dos : entero
Inicio
    Repetir
        Leer (a)
    Hasta_que(1<= a ∧ a <= 16)
    Repetir
        Leer (b)
    Hasta_que(1<=b ∧ b <= 16)
    Repetir
        Leer (c)
    Hasta_que(1<= c ∧ c <= 16)
    Repetir
        Leer (d)
    Hasta_que(1<= d ∧ d <= 16)
    desde i ← 1 hasta 4
        En_caso ( i )
            1: num ← a
            2: num ← b
            3: num ← c
            4: num ← d
        fin_caso
        pot ← 1
        binar ← 0
        Mientras (num > 0) hacer
            r ← num mod 2
            binar ←binar + r * pot
            pot ← pot * 10
            num ← num div 2
        fin_mientras
        En_caso ( i )
            1 : a ← binar
            2 : b ← binar
            3 : c ← binar
            4 : d ←binar
        fin_caso
    fin_desde
    N1 ← 0
    N2 ← 0
    N3 ← 0
```

```
N4 ← 0
Desde i ← 1 hasta 4
    En_caso ( i )
        1: num ← a
        2: num ← b
        3: num ← c
        4: num ← d
    fin_caso
    N1 ←N1 * 10 + num div 1000
    num ←num mod 1000
    N2 ←N2 * 10 + num div 100
    num ←num mod 100
    N3 ←N3 * 10 + num div 10
    num ←num mod 10
    N4 ←N4 * 10 + num
fin_desde
Desde i ← 1 hasta 4
    En_caso ( i )
        1 : num ← N1
        2 : num ← N2
        3 : num ← N3
        4 : num ← N4
    fin_caso
    decimal ← 0
    dos ←1
    Repetir
        r ← num mod 10
        decimal ←decimal + r * dos
        dos ← dos * 2
        num ← num div 10
    Hasta_que (num = 0)
    En_caso ( i )
        1 : N1 ← decimal
        2 : N2 ← decimal
        3 : N3 ← decimal
        4 : N4 ← decimal
    fin_caso
fin_desde
  Escribir ("Los números son: ", N1, N2, N3, N4)
Fin
```

PRIMERA PRÁCTICA CALIFICADA - UNI CICLO 2001- I

Lima 19 de abril del 2001.

PROBLEMA 1

En un experimento de física se desea medir las temperaturas de n cuerpos y averiguar la mayor temperatura negativa y la menor temperatura positiva. Crear un algoritmo que permita llevar a cabo este experimento. **Solución*:***

```
Pseudocódigo Experimento_de_física
Var     primpos, primneg : lógico      ( * primpos: primer positivo * )
                n, i : entero          ( * primneg: primer negativo * )
            temp, minpos, maxneg: real
Inicio
  Repetir
     Leer (n)
  Hasta_que (n > 0)
  primpos ← V
  primneg ← V
  Desde i ← 1 hasta n
     Leer (temp)
     Si (temp ≥ 0) entonces
        Si (primpos) entonces
           minpos ←temp
           primpos ← F
        Sino
           Si (minpos > temp) entonces
              minpos ← temp
           fin_si
        fin_si
     Sino
        temp ← temp * (-1)
        Si (primneg) entonces
           maxneg ← temp
           primneg ← F
        Sino
           temp ← temp * (-1)
```

```
        Si (maxneg > temp ) entonces
            maxneg ← temp
        fin_si
    fin_si
fin_si
fin_desde
maxneg ← maxneg * (-1)
Si (primpos = V ) entonces
    Escribir ("No se registraron temperaturas positivas")
Sino
    Escribir ("La menor temperatura positiva registrada es: ", minpos)
fin_si
Si (primneg = V ) entonces
    Escribir ("No se registraron temperaturas negativas")
Sino
    Escribir ("La mayor temperatura negativa registrada es; ", maxneg)
fin_si
Fin
```

PROBLEMA 2

De un conjunto de n productos de n productos se conoce el precio y el tipo de producto (tipo A, tipo B o tipo C). Se pide crear un algoritmo que permita leer esta información y luego presentar por cada tipo de producto el precio promedio de los productos que pertenecen a dicho tipo. **Solución:**

```
Pseudocódigo Problema2
Var     n, i, contA, contB, contC : entero
        tipo : caracter
        promA, promB, promC, precio : real
Inicio
    Repetir
        Leer (n)
    Hasta_que (n > 0)
    promA ← 0
    promB ← 0
```

```
promC ← 0
contA ← 0
contB ← 0
contC ← 0
desde i ←1 hasta n
    Repetir
        Leer (precio)
    Hasta_que (precio > 0)
    Repetir
        Leer (tipo)
    Hasta_que (tipo = 'A' ∨ tipo = 'B' ∨ tipo = 'C')
    En_caso (tipo)
        'A' :   promA ← promA + precio
                contA ← contA + 1
        'B' :   promB ← promB + precio
                contB ← contB + 1
        'C' :   promC ← promC + precio
                contC ← contC + 1
    fin_caso
fin_desde
Si (contA > 0) entonces
    promA ← promA/contA
    Escribir ("El promedio de los productos del tipo A es : ", promA)
Sino
    Escribir ("No se ingresaron productos del tipo A")
fin_si
Si (contB > 0) entonces
    promB ← promB/contB
    Escribir ("El promedio de los productos del tipo B es : ", promB)
Sino
    Escribir ("No se ingresaron productos del tipo B")
fin_si
Si (contC > 0) entonces
    promC ← promC/contC
    Escribir ("El promedio de los productos del tipo C es : ", promC)
Sino
    Escribir ("No se ingresaron productos del tipo C")
fin_si
Fin
```

PROBLEMA 3

De un conjunto no determinado de niños, se desea leer las edades de los mismos y luego determinar la frecuencia (número de apariciones) de la menor edad. Crear un algoritmo que permita llevar a cabo esto. **Solución:**

```
Pseudocódigo Edades
Var     edad, cont, mín_edad: entero
        primero: lógico
        resp: carácter
Inicio
   cont ← 0
   primero ← V
  Repetir
    Repetir
       Leer (edad)
    Hasta_que (1≤ edad ∧ edad < 18)
    Si (primero) entonces
       mín_edad ← edad
       cont ← 1
       primero ← F
    Sino
      Si (mín_edad = edad) entonces
         cont ← cont + 1
      Sino
        Si (mín_edad > edad) entonces
           mín_edad ← edad
           cont ← 1
        fin_si
      fin_si
    fin_si
    Escribir ("¿Desea seguir ingresando datos? (S/N) " )
    Leer (resp)
  Hasta_que (resp = 'N')
   Escribir ("La menor edad fue ", mín_edad, " y apareció ", cont, " veces ")
Fin
```

PRIMERA PRÁCTICA CALIFICADA - UNI CICLO 2005- I

Lima 19 de mayo del 2005.

PROBLEMA 1

Diseñe un algoritmo que permita ingresar un número entero positivo de por lo menos 10 cifras (N) y un dígito (d). (Nota: Usted debe validar los datos). Luego realice lo siguiente:

a) Presente el número de veces que aparece el dígito d y el número M que se forma con la secuencia de aparición de los dígitos d.

b) Muestre la suma (S) de los dígitos del número M así como el número invertido de S (SI)

Ejemplo:

Se ingresa: N=12345667896

Se ingresa: d=6

a) Número de apariciones de 6 es 3, M=666 *b) S=18; SI=81*

Solución*:*

Pseudocódigo Problema1

Var N, *d*, *aux1*, *contd*, M, S, r, *aux2*, SI: *entero*

Inicio

```
Repetir
    Leer (N)
Hasta_que (1000000000 ≤ N)
Repetir
    Leer (d)
Hasta_que (1 ≤ d ∧ d ≤ 9)
contd ← 0
aux1 ← N            ( * Para no perder el valor de N * )
M ← 0
Mientras (aux1 > 0) hacer
    r ← aux1 mod 10
    aux1 ← aux1 div 10
    Si (r = d) entonces
        contd ← contd + 1
        M ← M * 10 + d
    fin_si
fin_mientras
S ← d * contd       ( * Calculando la suma de los dígitos repetidos * )
```

```
SI ← 0
aux2 ← S            ( * Para no perder el valor de S * )
Mientras (aux2>0) hacer
    r ← aux2 mod 10
    aux2 ← aux2 div 10
    SI ← SI * 10 + r
fin_mientras
Escribir ("El número de veces que aparece el dígito ", d, " es = ", contd)
Escribir ("Secuencia formada M = ", M)
Escribir ("Suma de los dígitos S = ", S)
Escribir ("Suma invertida SI = ", SI)
Fin
```

PROBLEMA 2

Para seleccionar naranjas se cuenta con las siguientes categorías: A, B, C, D (A es de primera, B de segunda, C de tercera, D de cuarta calidad).

Diseñar un algoritmo que permita leer las categorías de n naranjas y luego presente la cantidad de naranjas por cada categoría ordenadas en forma decreciente.

Ejemplo:

Si n=1000 naranjas y se obtiene:	*La salida debe ser:*	
Categoría A: 300 naranjas	*B*	*500*
Categoría B: 500 naranjas	*A*	*300*
Categoría C: 50 naranjas	*D*	*150*
Categoría D: 150 naranjas	*C*	*50*

Solución*:*

Usaremos los acumuladores: SA, SB, SC y SD para almacenar en un primer momento el número de naranjas perteneciente a las categorías A, B, C y D respectivamente. También se usarán las variables carácter siguientes: C1, C2, C3, C4 para que almacenen el carácter que indique la categoría de las naranjas.

Pseudocódigo Naranjas

Var *i, n, nA, nB, nC, nD, aux*: entero

cat, C1, C2, C3, C4, auxc: caracter

Inicio

```
nA ← 0
nB ← 0
nC ← 0
nD ← 0
C1 ← 'A'
C2 ← 'B'
C3 ← 'C'
C4 ← 'D'
Repetir
    Leer (n)
Hasta_que (1<=n)  (*como mínimo se debe ingresar 1 naranja*)
Desde i ← 1 hasta n
    Repetir
        Escribir ("Ingrese categoría")
        Leer (cat)
    Hasta_que ('A'<= cat ∧ cat <= 'D')
    En_caso (cat)
        'A' : nA ← nA + 1
        'B' : nB ← nB + 1
        'C' : nC ← nC+ 1
        'D' : nD ← nD + 1
    fin_caso
fin_desde
Si (nA < nD) entonces
    aux ← SA
    SA ← SD
    SD ← aux
    auxc ← C1
    C1 ← C4
    C4 ← auxc
fin_si

Si (nB < nC) entonces
    aux ← nB
    nB ← nC
    nC ← aux
    auxc ← C2
    C2 ← C3
```

```
        C3 ← auxc
    fin_si

    Si (nA < nB) entonces
        aux ← nA
        nA ← nB
        nB ← aux
        auxc ← C1
        C1 ← C2
        C2 ← auxc
    fin_si

    Si (nC < nD) entonces
        aux ← nC
        nC ← nD
        nD ← aux
        auxc ← C3
        C3 ← C4
        C4 ← auxC
    fin_si

    Si (nB < nC) entonces
        aux ← nB
        nB ← nC
        nC ← aux
        auxc ← C2
        C2 ← C3
        C3 ← auxc
    fin_si

    Escribir (C1, nA)
    Escribir (C2, nB)
    Escribir (C3, nC)
    Escribir (C4, nD)
Fin
```

PROBLEMA 3

Se tienen las notas de la Primera Práctica Calificada de los alumnos de todas las secciones del curso ST 221 (secciones: U, V, W, X, Y, Z). Diseñar un algoritmo que permita ingresar la nota y la sección de los n alumnos del curso y luego presente:

a) La nota más alta y la nota promedio por cada sección.

b) El porcentaje total de aprobados (todas las secciones)

Solución*:*

Pseudocódigo Problema_3

Var *n, cont_ap, max_nota, i, nota, cont_U, cont_V, cont_W, cont_X, cont_Y, cont_Z* : entero
sum_U, sum_V, sum_W, sum_X, sum_Y, sum_Z: entero
sec: caracter

Inicio

Repetir
Leer (*n*)
Hasta_que (1 ≤ *n*) **(*como mínimo se debe ingresar 1 alumno*)**

cont_ap ← 0
max_nota ← 0
cont_U ← 0 : *sum_U* ← 0
cont_V ← 0 : *sum_V* ← 0
cont_W ← 0 : *sum_W*← 0
cont_X ← 0 : *sum_X* ← 0
cont_Y ← 0 : *sum_Y* ← 0
cont_Z ← 0 : *sum_Z* ← 0

Desde *i* ← 1 *hasta n*
Repetir
Leer (*nota*)
Hasta_que (0 ≤ *nota* ∧ *nota* ≤ 20)
Si (*nota* ≥ 10) **entonces**
cont_ap ← *cont_ap* + 1
fin_si
Si (*maxnota* < *nota*) **entonces**
maxnota ← *nota*
fin_si
Repetir
Leer (*sec*)
Hasta_que ('U'≤ *sec* ∧ *sec* ≤'Z')

```
    En_caso (sec)
        'U' : cont_U ← cont_U + 1
              sum_U ←sum_U + nota
        'V' : cont_V ← cont_V + 1
              sum_V ← sum_V + nota
        'W' : cont_W ← cont_W + 1
              sum_W ← sum_W + nota
        'X' : cont_X ← cont_X + 1
              sum_X ← sum_X + nota
        'Y' : cont_Y ← cont_Y + 1
              sum_Y ← sum_Y + nota
        'Z' : cont_Z ← cont_Z + 1
              sum_Z ← sum_Z + nota
    fin_caso
fin_desde
Escribir ("Nota más alta: ", maxnota)
Escribir ("Sección U: ", sum_U/cont_U)
Escribir ("Sección V: ", sum_V/cont_V)
Escribir ("Sección W: ", sum_W/cont_W)
Escribir ("Sección X: ", sum_X/cont_X)
Escribir ("Sección Y: ", sum_Y/cont_Y)
Escribir ("Sección Z: ", sum_Z/cont_Z)
Escribir ("Porcentaje total de aprobados: ", (cont_ap/n)*100 , "%")
Fin
```