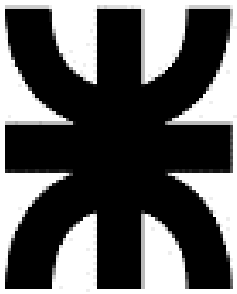
Seminario Universitario

2021



FRMDP

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

Formalización de Algoritmos

Docentes:

Gonzalo Benoffi
Matías Pascual
Guillermina Latorre
Melina Garcia
Emmanuel Etcheber
Gustavo Sonvico

INDICE DE CONTENIDO

Al estudiante:

El desarrollo de algoritmos es un tema fundamental en el diseño de programas por lo cual el alumno debe tener buenas bases que le sirvan para poder desarrollar de manera fácil y rápida sus programas.

Estos apuntes servirán de apoyo a los alumnos del Curso introductorio de las Carreras de Sistemas de la U.T.N., y al estudiante le facilitará desarrollar su capacidad analítica y creadora, para de esta manera mejorar su destreza en la elaboración de algoritmos que sirven como base para la codificación de los diferentes programas que tendrá que desarrollar a lo largo de su carrera.

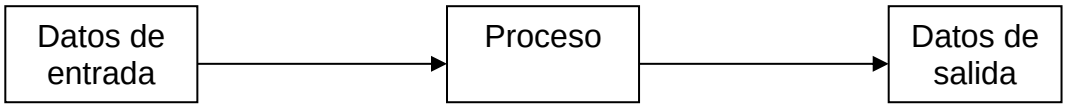
CAPÍTULO I
CONCEPTOS GENERALES

1.1 Introducción

La computadora no solamente es una máquina que puede realizar procesos para darnos resultados, sin que tengamos la noción exacta de las operaciones que realiza para llegar a esos resultados. Con la computadora además de lo anterior también podemos diseñar soluciones a la medida, de problemas específicos que se nos presenten. Más aún, si estos involucran operaciones matemáticas complejas y/o repetitivas, o requieren del manejo de un volumen muy grande de datos.
El diseño de soluciones a la medida de nuestros problemas, requiere como en otras disciplinas una metodología que nos enseñe de manera gradual, la forma de llegar a estas soluciones.
A las soluciones creadas por computadora se les conoce como programas y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, con un grupo de datos específicos. Lo anterior nos lleva al razonamiento de que un programa nos sirve para solucionar un problema específico.
Para poder realizar programas, además de conocer la metodología mencionada, también debemos de conocer, de manera específica las funciones que puede realizar la computadora y las formas en que se pueden manejar los elementos que hay en la misma.

Computadora: Es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los datos y la información se pueden introducir en la computadora como entrada (input) y a continuación se procesan para producir una salida (output).

Proceso de información en la computadora



Programa: Es el conjunto de instrucciones escritas en algún lenguaje de programación y que ejecutadas secuencialmente resuelven un problema específico.

1.2 Definiciones

Lenguaje*:*
Es una serie de símbolos que sirven para transmitir uno o más mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como comunicación.

Comunicación:
Es un proceso complejo que requiere una serie de reglas simples, pero indispensables para poderse llevar a cabo. Las dos principales son las siguientes:

- Los mensajes deben correr en un sentido a la vez.
- Debe forzosamente existir 4 elementos: Emisor, Receptor, Medio de Comunicación y Mensaje.

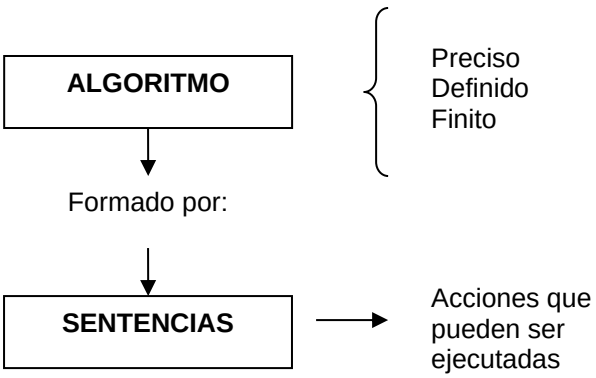
Lenguajes de Programación:
Es un conjunto de símbolos, caracteres y reglas (programas) que le permiten a las personas comunicarse con la computadora.
Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, calculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación.

1.3 Definición de Algoritmo

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.
Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.
Es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

- **Preciso:** no se presta a interpretaciones ambiguas
- **Definido:** si se siguen 2 o más veces los pasos, se obtiene el mismo resultado cada vez
- **Finito:** tiene comienzo y fin; tiene un número determinado de pasos.

Son ejemplos de algoritmos las instrucciones para hacer una receta de cocina, para usar un electrodoméstico, etc.
Los algoritmos se pueden expresar en forma de diagramas, por fórmulas y en PSEUDOCÓDIGO. Esta última herramienta es la más usada en lenguajes estructurados (como lenguaje C).



1.4 Tipos de Algoritmos

- **Cualitativos:** Son aquellos en los que se describen los pasos utilizando palabras.
- **Cuantitativos:** Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

1.5 Lenguajes Algorítmicos. Pseudocódigo

Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un algoritmo. En esencial, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.
Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un Pseudocódigo

- Permite representar en forma fácil operaciones repetitivas complejas
- Es muy fácil pasar el pseudocódigo de un programa a algún lenguaje de programación.
- Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

Diseño del Algoritmo

Las características de un buen algoritmo son:

- Debe tener un punto particular de inicio.
- Debe ser definido, no debe permitir dobles interpretaciones.
- Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.
- Existen varias herramientas para hacerlo, entre ellas pseudocódigo.

CAPITULO II
DESARROLLO DE ALGORITMOS

2.1 Tipos De Datos

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'b', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.

2.2 Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.
Por ejemplo: a + (b + 3)/c

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.
Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

- Aritméticas
- Relaciónales
- Lógicas

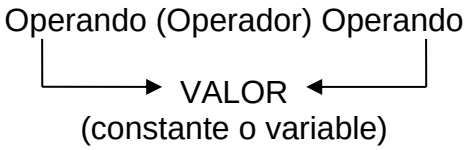
2.3 Operadores y Operandos

- **Operadores*:*** Son elementos que relacionan de forma diferente, los valores de una o más variables y/o constantes. Es decir, los operadores nos permiten manipular valores.

OPERADORES { Aritméticos, Relaciónales, Lógicos

2.3.1 Operadores Aritméticos.

Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).
Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.



Operadores Aritméticos

+	Suma
-	Resta
*	Multiplicación
/	División
Mod	Modulo (residuo de la división entera)

Ejemplos:

Expresión	Resultado
7 * 2	14
12 mod 7	5
4 + 2 * 5	14

Prioridad de los Operadores Aritméticos

- Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero.
- Dentro de una misma expresión los operadores se evalúan en el siguiente orden.
    1. Exponenciación
    2. Multiplicación, división, modulo.
    3. Suma y resta.
- Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

2.3.2 Operadores Relaciónales:

- Se utilizan para establecer una relación entre dos valores.
- Compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).
- Los operadores relaciónales comparan valores del mismo tipo (numéricos o cadenas)
- Tienen el mismo nivel de prioridad en su evaluación.
- Los operadores relaciónales tiene menor prioridad que los aritméticos.

Operadores Relaciónales

>	Mayor que
<	Menor que
> =	Mayor o igual que
< =	Menor o igual que
< >	Diferente
=	Igual

Ejemplos:
Si a = 10, b = 20 y c = 30

a + b > c	Falso
a - b < c	Verdadero
a - b = c	Falso
a * b < > c	Verdadero

2.3.3 Operadores Lógicos:

- Estos operadores se utilizan para establecer relaciones entre valores lógicos.
- Estos valores pueden ser resultado de una expresión relacional.

Operadores Lógicos	
And	Y
Or	O
Not	Negación

Operador AND		
Operando 1	Operando 2	Resultado
V	V	**V**
V	F	**F**
F	V	**F**
F	F	**F**

Operador OR		
Operando 1	Operando 2	Resultado
V	V	**V**
V	F	**V**
F	V	**V**
F	F	**F**

Operador NOT	
Operando	Resultado
V	**F**
F	**V**

Ejemplos:

Si a = 10, b = 20 y c = 30
(a < b) and (b < c)
V and V

Prioridad de los Operadores Lógicos

Not
And
Or

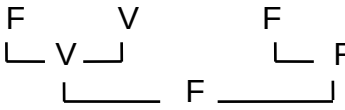
Prioridad de los Operadores en General

1.- ( )
2.- ^
3.- *, /, Mod, Not
4.- +, -, And
5.- >, <, > =, < =, < >, =, Or

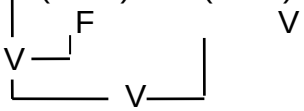
Ejemplos:

Si a = 10, b = 12, c = 13 y d =10

1) ((a > b)or(a < c)) and ((a = c) or (a > = b))



2) not (a = c) and (c > b)



2.4 Identificadores

Los identificadores representan los datos de un programa (constantes, variables, tipos de datos). Un identificador es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, que nos permite acceder a su contenido.

Ejemplo: nombre
num_hrs
calif2

Reglas para formar un Identificador

✓ Debe comenzar con una letra (A a Z, mayúsculas o minúsculas) y no deben contener espacios en blanco.
✓ Letras, dígitos y el carácter guion (-) están permitidos después del primer carácter.
✓ La longitud de identificadores. El nombre de un identificador debe ser descriptivo de aquello que representa, y considerando además la practicidad de su invocación durante el desarrollo del algoritmo. Si es muy extenso estaremos más expuestos a errores al nombrarlo.

2.5 Constantes y Variables

2.5.1 Variables.

Son objetos de un programa cuyo valor puede cambiar durante la ejecución del mismo. Datos que pueden sufrir modificaciones a lo largo de un programa. El cambio se produce mediante sentencias ejecutables como por ejemplo la asignación o el ingreso de datos. Es en realidad una porción (o posición) de memoria con nombre que permite almacenar temporalmente un dato durante la ejecución de un proceso. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo (a esto se lo denomina “declaración de variables”). Al declararlas se reserva una posición de memoria para la misma, donde se almacenará el valor que toma cada variable en cada momento del programa. El nombre de la posición es el NOMBRE DE LA VARIABLE y el valor almacenado es el VALOR DE LA VARIABLE. Las variables pueden ser de todos los tipos de datos conocidos: entero, decimal, carácter, cadena de carácter.

Variables y constantes se unen por medio de los operadores aritméticos, relacionales y lógicos constituyendo lo que se denomina una EXPRESIÓN.

2.5.2 Constante

Dato invariable a lo largo del programa. Es un valor que no puede cambiar durante la ejecución del programa; recibe un valor en el momento de la compilación del programa y este valor no puede ser modificado.

- Literales: es un valor de cualquier tipo que se utiliza como tal
  Ejemplo:
  sup-triangulo ← base * altura * 1/2
  1 y 2 son constantes literales

  numéricas: 5 , 3.14 , 4/3
  carácter: ‘S’ , ‘N’
  cadena: “Positivo”

Nota: las constantes de carácter se escriben entre apóstrofes o comillas simples y las de cadena entre comillas. De este modo se las diferencia de los nombre de las variables.

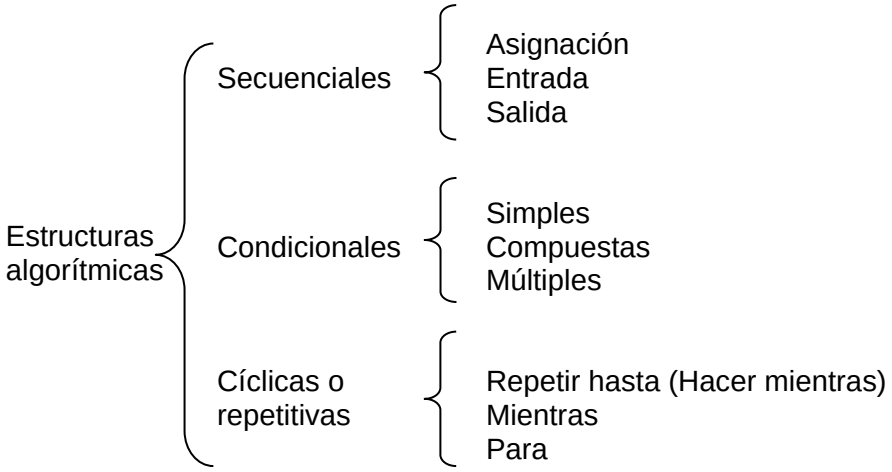
- Con nombre o declaradas: Se les asigna un nombre y un valor y no se lo modifica durante el transcurso del programa. En C se acostumbra nombrar esta clase de constantes con mayúsculas
  Ejemplo:
  Define constante PI valor 3.14
  sup-circulo ← PI * r * r

El uso de constantes literales limita la flexibilidad del programa. Es conveniente el uso de constantes con nombre.

- Tipeadas: Nos referimos a los textos que aparecen como mensajes de salida en un programa. Dichos mensajes son inalterables. Ejemplo: Mostrar ("Hola")

CAPÍTULO III
ESTRUCTURAS ALGORITMICAS

Las estructuras de operación de programas son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución de problemas. Estas estructuras se clasifican de acuerdo con su complejidad en:



3.1. Estructuras Secuenciales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. Una estructura secuencial se representa de la siguiente forma:

```
Comienzo
        Acción 1
        Acción 2
        .
        .
        Acción N
Fin
```

```
COMIENZO
        Levantar el teléfono
        Esperar tono
        Marcar número
        Esperar que contesten
        Hablar
        Colgar
FIN
```

3.1.1 Asignación

La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor.

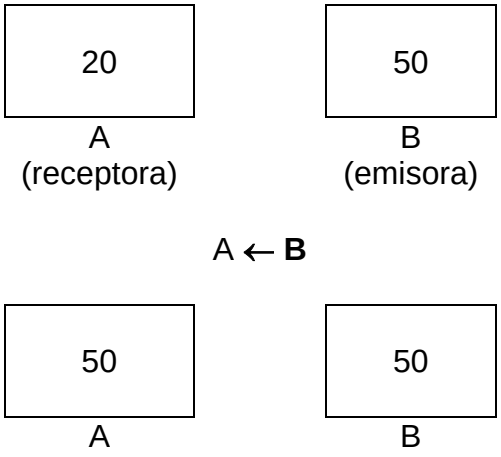
variable ← expresión

Puede ser una variable, una constante, una expresión o fórmula a evaluar

Ejemplos:

promedio ← suma /5
alumno ← nombre
cantidadDeNotas ← 5

Ejemplo de funcionamiento de la asignación:
En las variables transfiero el valor de B a A



ASIGNACIÓN: la emisora mantiene su valor y la receptora lo modifica, el dato de la emisora reemplaza al valor anterior

Se pueden asignar operaciones Ej. A← B * 2

La asignación se puede clasificar de la siguiente forma:

- ✓ **Simples:** Consiste en pasar un valor constante a una variable (a←15)
- ✓ **Contador:** Es una variable que se incrementa, cuando se ejecuta, en una unidad o en una cantidad constante

contador ← contador + 1
múltiplo ← múltiplo + 3

- ✓ **Acumulador:** Es una variable que se incrementa en una cantidad variable

suma ← suma + numero

Donde “numero” es una variable que recibe distintos valores

Considerar:

- ✓ Una variable del lado derecho debe tener valor antes de que la sentencia se ejecute.
  Si “numero” no tiene valor antes de: suma ← suma + numero
  Se produce un Error LÓGICO. Se dice que “numero” no se ha inicializado.

- ✓ A la izquierda de una sentencia de asignación sólo puede haber variables. No puede haber operaciones.

Nota: la operación de asignación es una **operación destructiva** debido a que el valor almacenado en una variable se pierde o destruye y se sustituye por el nuevo valor de asignación. Ejemplo
numero ← 16
numero ← -23
numero conservará el último valor asignado, en este caso -23

3.1.2 Entrada
La entrada de datos consiste en recibir desde un dispositivo de entrada (p.ej. el teclado) un valor.
Esta operación se representa en pseudocódigo como sigue:

```
Leer (a)
Leer (b)
```

Donde “a” y “b” son las variables que recibirán los valores

3.1.3 Salida
Consiste en mandar por un dispositivo de salida (p.ej. monitor o impresora) un resultado o mensaje.
Este proceso se representa en pseudocódigo como sigue:

```
Mostrar ( “El resultado es:”, R )
```

Donde “El resultado es:” es un mensaje que se desea aparezca y R es una variable que contiene un valor.

3.2 Estructura Condicional
Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que, en base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen dos tipos básicos, las simples y las múltiples.

3.2.1 Simples
Las estructuras condicionales simples se las conoce como “Tomas de decisión”. Estas tomas de decisión tienen la siguiente forma:

```
Si <condición> entonces
        Acciones
Fin-si
```

3.2.2. Dobles
Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:

```
Si <condición>  entonces
        Acciones
Sino
        Acciones
Fin-si
```

Donde:

Si	Indica el comando de comparación
Condición	Indica la condición a evaluar
Entonces	Precede a las acciones a realizar cuando se cumple la condición
Acciones	Son las acciones a realizar cuando se cumple o no la condición
Sino	Precede a las acciones a realizar cuando no se cumple la condición

Dependiendo de si la comparación es cierta o falsa, se pueden realizar una o más acciones.

3.2.3 Múltiples

Las estructuras de comparación múltiples, son tomas de decisión especializadas que permiten comparar una variable contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

```
LEER (variable)
CON-SELECCIÓN (VARIABLE) HACER
        CASO constante1:
                        Sentencias
                        ROMPER
        CASO constante2:
                        Sentencias
                        ROMPER
        CASO constanteN:
                        Sentencias
                        ROMPER
        OTROS CASOS:
                        Sentencias
FIN-SELECCIÓN
```

La estructura de selección múltiple sólo compara por igualdad el valor de la variable con cada una de las constantes de cada caso. Al encontrar una coincidencia comienza a ejecutar las sentencias en forma secuencial hasta encontrar el fin de la estructura o una instrucción que rompa la misma.
Puede tener hasta 257 casos.
No puede haber 2 casos con el mismo valor en la constante.
Sólo se pueden utilizar variables de tipo carácter o enteras.
Si la variable que se está seleccionando es de tipo carácter, las constantes de tipo carácter se colocan entre comillas simples o apóstrofes, para el caso de variables de tipo enteras, las constantes numéricas se colocan directamente.
Puede contener casos vacíos.

Ejemplo:

```
COMIENZO
   I ← 1
   MIENTRAS ( I < 7 ) HACER
        CON-SELECCIÓN ( I ) HACER
        CASO 2:
        CASO 4:
        CASO 6:
```

```
            MOSTRAR("I ES PAR")
            ROMPER
        CASO 1:
        CASO 3:
        CASO 5:
            MOSTRAR("I ES IMPAR")
            ROMPER
        OTROS CASOS:
            MOSTRAR("I VALE 0")
    FIN-SELECCIÓN
    I ← I + 1
  FIN-MIENTRAS
FIN
```

3.3. Estructuras Cíclicas o Repetitivas

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa).Los ciclos se clasifican en:

Ciclos con un número indeterminado de iteraciones (Mientras, Hacer-Mientras)

Son aquellos en que el número de iteraciones no se conoce con exactitud, ya que esta dado en función de un dato dentro del programa.

Mientras

Esta es una estructura que repetirá un proceso durante "N" veces, donde "N" puede ser fijo o variable. Para esto, la instrucción se vale de una condición que es la que debe cumplirse para que se siga ejecutando. Cuando la condición ya no se cumple, entonces ya no se ejecuta el proceso. La forma de esta estructura es la siguiente:

```
Mientras <condición> Hacer
    Accion1
    Accion2
      .
      .
    AccionN
Fin-mientras
```

Hacer - mientras

Esta es una estructura similar en algunas características, a la anterior. Repite un proceso una cantidad de veces, pero a diferencia del Mientras, el **Hacer - mientras** permite realizar el proceso cuando menos una vez, ya que la condición se evalúa al final del proceso, mientras que en el Mientras puede ser que nunca llegue a entrar si la condición no se cumple desde un principio. La forma de esta estructura es la siguiente:

```
HACER
        Accion1
        Accion2
           .
           .
        AccionN
MIENTRAS <condición>
```

Ciclos con un Número Determinado de Iteraciones (Para)

Son aquellos en que el número de iteraciones se conoce antes de ejecutarse el ciclo. La forma de esta estructura es la siguiente:

```
Para (variable ← desde  a hasta incremento) Hacer
        Accion1
        Accion2
       AccionN
Fin-para
```

Dónde:
Variable: Variable de control del ciclo (un contador)
Desde: Limite inferir
Hasta: Límite superior
Incremento: indica de qué modo se incrementa la variable de control (de 1 en 1, de 2 en 2, etc.)

En este ciclo la variable de control toma el valor inicial del ciclo y el ciclo se repite hasta que la variable de control llegue al límite superior. La cantidad de repeticiones que tenga depende del límite superior y del incremento de la variable

ANEXO - Convenciones de Pseudocódigo a utilizar

SENTENCIAS/ ESTRUCTURAS	PSEUDOCÓDIGO
ASIGNACIÓN	←
OPERADORES ARITMÉTICOS	**-** (resta) + (suma) / (división) * (multiplicación) **Mod** (resto de división entre enteros)
OPERADORES RELACIONALES	< (menor) ≤ (menor o igual) > (mayor) ≥ (mayor o igual) = (igual) < > (distinto)
OPERADORES LÓGICOS	Y (and) O (or) NOT (not)
ENTRADA / SALIDA	MOSTRAR **(**nombre de variable o constante**)** LEER **(**nombre de variable**)**
ESTRUCTURA SELECTIVA SIMPLE	SI (condición) ENTONCES Sentencias FINSI
ESTRUCTURA SELECTIVA COMPUESTA	SI (condición) ENTONCES Sentencias SINO Sentencias FINSI
ESTRUCTURAS REPETITIVAS	MIENTRAS (condición) HACER Sentencias FIN MIENTRAS HACER Sentencias MIENTRAS (condición)
ESTRUCTURA DE SELECCIÓN MÚLTIPLE	CON-SELECCIÓN (variable) HACER CASO constante1: Sentencias ROMPER CASO constante2: Sentencias ROMPER CASO constante3: Sentencias ROMPER OTROS-CASOS:

	Sentencias FIN-SELECCIÓN
MANEJO DE PANTALLA	BORRAR PANTALLA
OTRAS NOTAS	Las constantes de tipo carácter se colocan entre apóstrofes (comillas simples) y las constantes de cadena entre comillas. Es para diferenciar las constantes de las variables Para salir de una estructura repetitiva antes de que finalice (por su condición) o alcance el número de iteraciones indicados (en el para) utilizamos la sentencia ROMPER. Los comentarios los consignamos entre llaves { .... }

ANEXO - Buenas prácticas de programación

Antes de escribir los algoritmos de los ejercicios en Pseudocódigo debemos considerar que un programa legible y comprensible es más fácil de entender, corregir y mantener.
Algunas consideraciones:

1. **SANGRADO O INDENTACIÓN**
   En cada estructura, y alineando las instrucciones (sentencias) dentro de cada una de ellas y dentro de todo el algoritmo (Comienzo, Fin)

2. **LÍNEAS EN BLANCO**
   Dejarlas entre partes importantes o que estén lógicamente separadas. Recomendable luego de cada estructura (Repetitiva o Selectiva), luego de declaración de variables.

3. **COMENTARIOS**
   Parte importante de la documentación de un programa que permite mayor comprensión del mismo. (luego lo haremos en C)
   En Pseudocódigo entre **{ }**
4. **NOMBRES SIGNIFICATIVOS DE IDENTIFICADORES Y FUNCIONES**
   Nombres que representen aquello que estamos tratando. Si son palabras compuestas usar guion común. Todos deben comenzar con una letra y nunca con un número.
   La declaración de variables con nombres significativos ayuda a que los programas estén auto documentados, es decir, que resulte más fácil entenderlos simplemente leyéndolos en lugar de tener que consultar manuales o hacer referencia a demasiados comentarios.

5. **CADA SENTENCIA EN UNA LÍNEA DISTINTA**
   Al colocar una nueva sentencia comenzar una nueva línea, incluso las palabras claves de las estructuras en líneas separadas. En los programas no debe haber más que una instrucción por línea.

6. **ESPACIOS ENTRE ELEMENTOS DE UNA SENTENCIA**
   En aconsejable poner espacios en ambos lados de los operadores binarios. Con esto se resalta el operador y se simplifica la lectura del programa.

   Por ejemplo: suma ← suma + numero