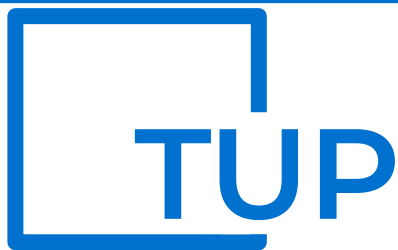


TECNICATURA  
UNIVERSITARIA  
EN PROGRAMACIÓN  
UTN-FRC



**UTN**  
Facultad Regional Córdoba

# TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

## PROGRAMACIÓN

Nivelación

Material Teórico



## Índice

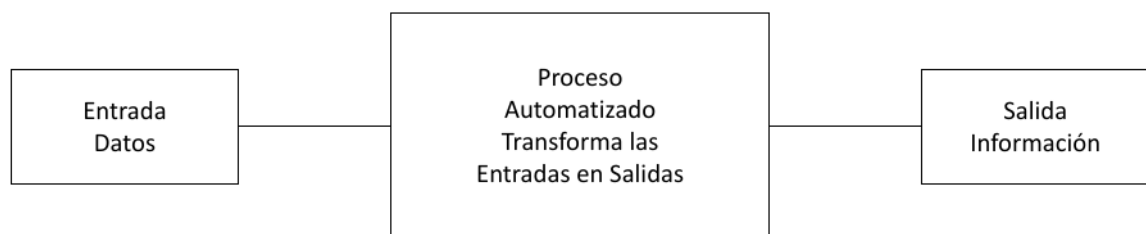
MODULO 1: RESOLUCIÓN DE PROBLEMAS POR COMPUTADORA.....	2
1. METODOLOGÍA .....	2
2. ETAPAS EN LA RESOLUCIÓN DE UN PROBLEMA .....	3
3. ALGORITMOS .....	4
4. DIAGRAMA DE FLUJO.....	5
5. PROGRAMAS.....	6
MODULO 2: PROGRAMACIÓN ESTRUCTURADA .....	8
1. DATOS.....	9
2. VARIABLES Y CONSTANTES .....	9
2.1 Expresiones .....	12
3. OPERADORES.....	13
4. LECTURA DE DATOS (ENTRADA).....	16
5. ESCRITURA DE RESULTADOS (SALIDA) .....	16
MODULO 3. ESTRUCTURAS SECUENCIALES .....	17
3.1 Diagrama de flujo.....	18
MODULO 4: Estructuras condicionales.....	19
1. ALTERNATIVA SIMPLE.....	19
2. ALTERNATIVA DOBLE.....	20
3. ALTERNATIVA MÚLTIPLE .....	20
MODULO 5: Estructuras repetitivas .....	24
1. ESTRUCTURA MIENTRAS .....	25
2. ESTRUCTURA REPETIR O HACER MIENTRAS .....	25
3. ESTRUCTURA DESDE / PARA.....	27
MODULO 6: Casos de estudio .....	34
Problema Modelo.....	34
BIBLIOGRAFÍA .....	38

**MODULO 1: RESOLUCIÓN DE PROBLEMAS POR COMPUTADORA****1. METODOLOGÍA**

La metodología de resolución de problemas por computadora se basa en las siguientes etapas secuenciales a seguir:

**1º Paso:** Lectura inicial y análisis a priori de la situación planteada en el problema.

**2º Paso:** Análisis detallado del problema basado en el Enfoque Sistémico, que lo delimita y lo divide en sub-problemas más simples para su comprensión (Diseño Top Down o Descendente). En este paso, es necesario destacar que las **salidas** de un sub problema pueden ser las **entradas** de otro, por lo que nunca debe perderse la visión holística de la solución (Concepto de Sistemas) respetando la siguiente secuencia de pasos a respetar:



**Gráfico 1:** Elaboración propia

**3º Paso:** Graficar el **Diagrama de Flujo**, que es la representación gráfica de la resolución del problema planteado (o **Algoritmo**), a través de la implementación del diseño Top Down (de lo general a lo particular), previamente analizado en la etapa anterior.

**4º Paso** Comprobar que el Algoritmo funciona para una muestra pequeña y representativa de la población, lo cual implica seleccionar aquellas entradas de datos que pueden presentarse; mediante el uso de la herramienta Prueba de Escritorio (matriz  $n \times n$  variables que facilita la ratificación o rectificación de la lógica implementada) y es una actividad áulica.

**5º Paso** Codificar el Algoritmo a un lenguaje natural denominado Pseudocódigo, como una instancia previa y de consolidación de la lógica empleada, tendiente a traducir cada acción del Diagrama de Flujo planteado a instrucciones.

**6º Paso** Codificar el Algoritmo en un lenguaje de programación y ejecutarlo, llevando el Algoritmo y Pseudocódigo realizado en los pasos anteriores; verificando que el mismo arroja los mismos resultados que la prueba de escritorio realizada en forma manual.

## 2. ETAPAS EN LA RESOLUCIÓN DE UN PROBLEMA

Las etapas o partes del proceso de Resolución de un problema mediante computadora son:

### a. Análisis del problema

Requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo el detalle.

El análisis del problema exige:

- Elección de la forma de encarar el problema y su resolución a través de un método adecuado, como por ejemplo: estructurado, orientado a objetos (POO), entre otros.
- Una vez elegida la forma de trabajo realizar una lectura previa del problema a fin de obtener una idea general de lo que se solicita.
- La segunda lectura deberá servir para responder a las preguntas: ¿Qué información debe proporcionar la resolución del problema? (Salidas) y ¿Qué datos se necesitan para resolver el problema? (Entradas)

### b. Diseño o desarrollo del algoritmo

Una vez analizado el problema, se debe desarrollar el algoritmo, procedimiento paso a paso para solucionar el problema dado. La descomposición del problema original en sub-problemas más simples y la división de estos en otros más simples se denomina diseño descendente (Top Down Design). Tras esta primera descripción, éstos se amplían en una descripción más detallada con más pasos específicos. Este proceso se denomina refinamiento del algoritmo.

Las ventajas más importantes del diseño descendente o Top Down (de lo general a lo particular) son:

- El problema (todo) se comprende más fácilmente al dividirse en partes más simples denominadas módulos (partes).
- Las modificaciones en los módulos son más fáciles.
- La comprobación del problema se puede verificar fácilmente.

Tras los pasos anteriores es preciso representar el algoritmo mediante una herramienta de programación: diagrama de flujo, pseudocódigo.

### c. Resolución del algoritmo en la computadora

Para resolver el algoritmo mediante una computadora se necesita codificar el algoritmo en un lenguaje de programación, convertir el algoritmo en programa, ejecutarlo y comprobar que el programa soluciona verdaderamente el problema.

Una vez que el algoritmo está diseñado y representado gráficamente mediante una herramienta de programación se debe pasar a la fase de resolución

práctica del problema con la computadora. Esta fase se descompone a su vez en las siguientes subfases:

- Codificación del algoritmo en un programa.
- Ejecución del programa.
- Comprobación del programa.

En el diseño del algoritmo, éste se describe en una herramienta de programación tal como un diagrama de flujo o pseudocódigo.

Sin embargo, el programa que implementa el algoritmo debe ser escrito en un lenguaje de programación y siguiendo las reglas gramaticales o sintaxis del mismo. La fase de conversión del algoritmo en un lenguaje de programación se denomina codificación, ya que el algoritmo en lenguaje de programación se denomina código.

### 3. ALGORITMOS

#### a. Concepto y características fundamentales

- Son un conjunto finito de reglas diseñadas para crear una secuencia de operaciones para resolver un tipo específico de problemas.
- Ecuación de Wirth: Algoritmos + Estructuras de Datos = Programas
- Una estrategia que exige precisión en las instrucciones (descripción de los pasos a seguir), donde cada instrucción debe ser clara y precisa para que no surja ninguna duda en cuanto a su ejecución, y sencilla para que no surja ninguna duda en su comprensión y pueda ser ejecutada sin dificultad.

#### b. Diseño Top Down

Significa descomponer un programa en términos de recursos abstractos, es decir consiste en descomponer una determinada acción compleja en términos de un número de acciones más simples capaz de ejecutarlas o constituyan instrucciones de computadora disponible.

Un programa estructurado cumple las siguientes características:

- Posee un solo punto de entrada y uno de salida o fin para control del programa.
- Existen caminos desde la entrada hasta la salida que se pueden seguir y que pasan por todas partes del programa.
- Todas las instrucciones son ejecutables y no existen lazos o bucles infinitos (sin fin); son finitas.

Los Algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y, ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizarán sin importar el cocinero.

#### c. Representación de los algoritmos

Los métodos para representar un algoritmo son:

- Diagrama de flujo
- Pseudocódigo
- Lenguaje español
- Fórmulas

## 4. DIAGRAMA DE FLUJO

Es una de las técnicas de representación de algoritmos más antigua y a la vez la más utilizada.

Un diagrama de flujo es un diagrama que utiliza un conjunto de símbolos estándar (cajas) para representar los pasos de un algoritmo. Los símbolos estándar normalizados son muy variados, sin embargo, los símbolos más utilizados representan:

- Proceso
- Decisión
- Fin
- Entrada / Salida
- Dirección del flujo

La siguiente tabla presenta los símbolos más utilizados en la confección de los diagramas de flujo.







Símbolo	Función
	<b>Terminal</b> representa el comienzo y el final de un programa
	<b>Entrada/Salida</b> , cualquier tipo de introducción de datos en la memoria desde los periféricos
	<b>Proceso</b> , cualquier tipo de operación que pueda originar cambio de valor, operaciones aritméticas, transferencia
	<b>Decisión</b> , indica operaciones lógicas o de comparación entre datos y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas
flechas	<b>Indicador de dirección o línea de flujo</b> , indica el sentido de ejecución de las operaciones
	<b>Impresora</b> , se utiliza en ocasiones en lugar del símbolo de E/S
	<b>Subrutina</b> , indica comienzo de una subrutina o función

Gráfico 2: Elaboración propia

## 5. PROGRAMAS

### a. Programación del Algoritmo

Para Wirth un Programa viene definido por la ecuación: Algoritmos + Estructuras de Datos = Programas.

Un **programa** consta de dos (2) componentes: una cabecera de programa y un bloque algoritmo.

La cabecera de programa es una acción simple que comienza con las declaraciones de variables y constantes que tengan nombres.

El bloque algoritmo es el resto del programa que consta de las acciones ejecutables. Éstas últimas son las acciones que posteriormente deberá realizar la computadora cuando el algoritmo sea convertido en un programa ejecutable.

Esta fase se descompone a su vez en las siguientes subfases:

- Codificación del algoritmo en un programa.
- Ejecución del programa.
- Comprobación del programa.

La fase de conversión del algoritmo en un lenguaje de programación se denomina codificación, ya que el algoritmo en lenguaje de programación se denomina código.

Un programa de computadora es un conjunto de instrucciones, órdenes dadas a la máquina, que producirán la ejecución de una determinada tarea. El proceso de diseñar un programa es, esencialmente, un proceso creativo, es un proceso de solución de problemas que requiere las siguientes fases:

1. Análisis del problema.
2. Diseño de algoritmo solución.
3. Codificación.
4. Compilación y ejecución.
5. Verificación.
6. Depuración.
7. Documentación.
8. Mantenimiento.



## MODULO 2: PROGRAMACIÓN ESTRUCTURADA

El paradigma de la programación estructurada plantea que todo problema puede resolverse con la combinación de tres estructuras básicas:

**Estructura secuencial:** en esta estructura todas las instrucciones son ejecutadas en forma secuencial y obligatoria, sin posibilidad de repetir o de seleccionar la ejecución o no de cada instrucción.

**Estructura condicional:** esta estructura permite identificar mediante una condición si una instrucción o conjunto deben ser ejecutadas. Por ejemplo, si se requiere calcular el cociente entre dos números, es necesario verificar que el denominador sea distinto de 0 y únicamente en ese caso realizar el cálculo.

**Estructura repetitiva:** Las estructuras repetitivas permiten repetir la ejecución de una instrucción o conjunto, estableciendo un ciclo o bucle que se ve interrumpido al alcanzar cierta condición que indique que deben finalizarse las repeticiones.

## 1. DATOS

Los datos pueden definirse como piezas de información con las que un programa trabaja. Cada dato tiene asociado un tipo. El tipo de dato determina la naturaleza del conjunto de valores que un dato puede tomar. Los tipos de datos que se utiliza en los programas, en general, son los siguientes:

- Datos carácter. El tipo carácter es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación. La longitud de una cadena de caracteres es el número de ellos comprendidos entre los separadores o delimitadores. Los caracteres que reconocen las diferentes computadoras no son estándar; sin embargo, la mayoría reconoce los caracteres alfabéticos (A, B, C, ..., Z), numéricos (0 al 9) y especiales (+, -, %, ..., \$).
- Datos numéricos. El tipo numérico es el conjunto de los valores numéricos. Estos pueden representarse en dos formas distintas:
  - Enteros: Los enteros son números completos, no tienen componentes fraccionarios o decimales y pueden ser negativos o positivos. Ejemplos de números enteros son: 5 6 -7 20
  - Reales: Los números reales siempre tienen un punto decimal y pueden ser positivos o negativos. Un número real consta de un entero u una parte decimal. Los siguientes números son reales: 0.08 3.154 -12.5
- Datos lógicos (booleanos). El tipo lógico, también denominado booleano, es aquel dato que sólo puede tomar uno de dos valores: verdadero (true) o falso (false). Este tipo de dato se utiliza para representar las alternativas (si/no) a determinadas condiciones. Por ejemplo, cuando se pide si un valor entero es par, la respuesta será verdadera o falsa, según sea par o impar.

## 2. VARIABLES Y CONSTANTES

**Variables.** Una variable es una unidad de almacenamiento identificada por un nombre y un tipo de dato, tiene una dirección (ubicación) dentro de la memoria y puede almacenar información según el tipo de dato que tenga asociado. Si imaginamos la memoria del computador como un conjunto de celdas, entonces podemos decir que una variable es una de esas celdas que el usuario identifica con un nombre. Es necesario definir las variables de un programa antes de que puedan ser utilizadas. Los identificadores son nombres simbólicos que se asignan a las celdas de memoria en lugar de su dirección numérica.

**Constantes.** Los programas de computadora contienen ciertos valores que no deben cambiar durante la ejecución del programa. Tales valores se llaman constantes.

#### Tipos de constantes

La mayoría de los lenguajes de programación permiten diferentes tipos de constantes: enteras, reales, caracteres y lógicas, y representan datos de esos tipos.

Constantes reales: 1.234 -0.1436 -1.5443724

Una constante tipo carácter o constante de caracteres consiste en un carácter válido encerrado dentro de comillas, por ejemplo: "B" "+" "3.14"

Constantes lógicas, sólo existen dos constantes lógicas o booleanas: Verdadero Falso

#### Características de las variables

Una variable es un objeto o partida de datos cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa. Dependiendo del lenguaje, hay diferentes tipos de variables tales como enteras, reales, carácter lógicas y de cadena.



**Gráfico 3:** Elaboración propia

Una variable que es de un cierto tipo puede tomar únicamente valores de ese tipo. Una variable de carácter, por ejemplo, puede tomar como valor sólo caracteres, mientras que una variable entera puede tomar sólo valores enteros. Si se intenta asignar un valor de un tipo a una variable de otro tipo se producirá un error de tipo.

Una variable se identifica por los siguientes atributos: nombre que lo asigna y tipo que describe el uso de la variable.

Los nombres de las variables o identificadores, suelen constar de varios caracteres alfanuméricos de los cuales el primero normalmente es una letra. No se debe utilizar los nombres de identificadores o palabras reservadas del lenguaje de programación.

Algunos nombres válidos de variables son: A510 NOTAS  
NOMBRE\_APELLIDO PROMEDIO

Los nombres de las variables elegidas para el algoritmo o el programa deben ser significativos y tener relación con el objeto que representan, como pueden ser los casos siguientes:

NOMBRES para representar nombres de personas NOTAS para representar las notas de una clase PRECIO para representar los precios de diferentes artículos.

a. Variables Especiales

**Contadores**

Es una variable que, a partir de un valor inicial, aumenta o disminuye en una cantidad constante. La forma general de uso es:

contador = valor inicial

contador = contador + paso

Donde paso es el valor constante que se suma o resta en cada repetición.

Por ejemplo:

c = 0

c = c + 1

d = 120

d = d + 10

Los procesos repetitivos son la base del uso de las computadoras. En estos procesos se necesitan normalmente contar los sucesos o acciones internas del bucle, como pueden ser los elementos de un fichero, el número de iteraciones a realizar por el bucle, etc. Una forma de controlar un bucle es mediante un contador.

**Acumuladores**

Es una variable que, a partir de un valor inicial, aumenta o disminuye en una cantidad no constante. Se utilizan para sumar o restar variables afines.

acumulador = valor inicial

acumulador = acumulador + cantidad

En donde cantidad es la variable que se suma o resta en cada repetición, por ejemplo:

a = 0

cant = 120

a = a + cant

Un acumulador es una variable cuya misión es almacenar cantidades variables resultantes de sumas sucesivas. Realiza la misma función que un contador con la diferencia de que el incremento o decremento de cada suma es variable en lugar de constante como en el caso del contador.

**Banderas o interruptores**

Una bandera, interruptor o indicador es una variable que puede tomar diversos valores a lo largo de la ejecución del programa y que permite comunicar información de una parte a otra del mismo. Por lo general toman dos valores ( ej. 0 y 1) y se utilizan para saber si se ha ejecutado o no una parte del proceso. De acuerdo al valor que tenga en el momento en que se le testea se puede determinar si se ejecuta o no esa parte del proceso.

- Definir variables, luego definir constantes. Tipificar constantes y luego en un título aparte indicar las características de las variables.
- Otra alternativa es definir Variables y dar todas las características y tipos especiales; y luego describir las CONSTANTES y los tipos que existen.

## 2.1 Expresiones

La combinación de operadores y operandos da lugar a lo que se denominan expresiones. Una expresión es evaluada por el lenguaje dando lugar principalmente a dos tipos de resultados, basándose en los cuales podemos dividir las expresiones en aritméticas y condicionales. Las primeras se caracterizan por generar un resultado numérico, mientras que las segundas sólo pueden devolver como resultado los valores verdadero (true) o falso (false).

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombre de funciones especiales. Las mismas ideas son utilizadas en notación matemática tradicional; por ejemplo:

$$a + b * (b + 3) + \left(\frac{a}{10}\right)$$

Cada expresión toma un valor que se determina evaluando las variables y constantes implicadas junto con la ejecución de las operaciones indicadas.

Según sea el tipo de objetos que manipulan, se clasifican las expresiones en:

- Aritméticas
- Lógicas

### a. Expresiones Aritméticas

Las expresiones aritméticas son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas (real o enteras) y las operaciones son las aritméticas. Los símbolos +, -, \*, / representan a los operadores aritméticos.

En la expresión “5 + 3” los valores 5 y 3 se denominan operandos. El valor de la expresión 5 + 3 se conoce como resultado de la expresión.

Los operadores se utilizan de igual forma que en matemática. Por consiguiente, A x B se escribe en un algoritmo como A \* B y 1/4 x C como C/4. al igual que en matemática el signo menos juega un doble papel, como resta en A - B y como negación en -A.

Los cálculos que implican tipos de datos reales y enteros suelen dar normalmente resultados del mismo tipo si los operandos lo son también. Por ejemplo, el producto de operandos reales produce un real.

Operador	Significado	Tipo de operandos	Tipo de resultado
+	Suma	Entero o Real	Entero o Real
-	Resta	Entero o Real	Entero o Real
*	Multiplicación	Entero o Real	Entero o Real
/	División	Entero o Real	Entero o Real
	Modulo o resto de división		
%	entera	Entero	Entero

Tabla 1: Elaboración propia

**Regla de prioridad:** Cuando las expresiones tienen dos o más **operadores** requieren ciertas reglas que permitan determinar el orden de las operaciones asociadas. Dichas reglas se denominan reglas de prioridad o precedencia y son: Las operaciones que están encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis uno dentro del otro, las expresiones más internas se evalúan primero. Las operaciones aritméticas dentro de una expresión suelen seguir el orden de prioridad similar al de la matemática, es decir, primero se evalúan la multiplicación y la división y luego las sumas y restas.

#### b. Expresiones lógicas (booleanas)

Son aquellas cuyo valor es siempre verdadero o falso. Recuerde que existen dos constantes lógicas: verdadera y falsa, que las variables lógicas pueden asumir como valores posibles. En esencia, una expresión lógica es una expresión que sólo puede tomar uno de estos dos valores. Se denomina también expresiones booleanas en honor del matemático británico George Boole, que desarrolló el Álgebra lógica o de Boole.

Las expresiones lógicas se forman combinando constantes lógicas, variables lógicas y otras expresiones lógicas utilizando los operadores lógicos: **not**, **and** y **or**, y los operadores relacionales (o de comparación): **=**, **<**, **>**, **<=**, **>=**, **<>**.

Cuando se relacionan varias expresiones lógicas, el resultado de esa relación implica un único valor de verdad. Es por eso que cuando las expresiones son más complejas y las relaciones más numerosas se utilizan la representación gráfica a través de tablas de verdad.

### 3. OPERADORES

#### a. Operadores de Relación

Los operadores relacionales o de relación permiten realizar comparaciones de valores de tipo numéricos, carácter o lógico.

Los operadores de relación sirven para expresar las condiciones en los algoritmos.

Operador	Significado
<	Menor que
>	Mayor que
=	Igual que
<=	Menor o igual a
>=	Mayor o igual a
!=	Distinto de

**Tabla 2:** Elaboración propia

El formato general para las comparaciones es: [Expresión 1 operador de relación expresión 2], y el resultado de la operación será verdadero o falso. Así, por ejemplo

si A = 4 y B = 3 entonces  
A > B es verdadero  
(A -2) < (B-4) es falso

Para realizar comparaciones de datos tipo carácter, se requiere una secuencia de ordenación de los caracteres, similar al orden creciente o decreciente. Esta ordenación suele ser alfabética, tanto mayúscula como minúscula, y numérica, considerándolas de modo independiente, pero si se consideran caracteres mixtos, se debe recurrir a un código normalizado como es el ASCII.

Estos códigos normalizados son:

- Los caracteres especiales #, %, \$, (), etc.
- Los valores de los caracteres que representan a los dígitos están en su orden natural. Esto es "0" < "1", "1" < "2",
- Las letras mayúsculas A a Z siguen el orden alfabético. "A" < "B", "C" < "F", etc.
- Si existen letras minúsculas, éstas siguen el mismo criterio alfabético "a" < "b", "f" < "v", etc.

En general, los cuatro grupos anteriores están situados en el código ASCII en orden creciente. Así, "1" < "A" y "B" < "C". Sin embargo, para tener completa seguridad será preciso consultar el código de caracteres de su computadora.

Cuando se utilizan los operadores de relación, con valores lógicos, la constante false (falsa) es menor que la constante true (verdadera)

false < true  
true > false

Algunos lenguajes (por ejemplo C) consideran a la constante false con el valor 0 y a la constante true con un valor distinto de 0.



**b. Operadores Lógicos**

Los operadores lógicos, también llamados conectivos lógicos, son vínculos que actúan entre las expresiones lógicas o proposiciones simples y generan nuevas expresiones o proposiciones compuestas. Los operadores lógicos básicos son: not (no), and (y), or (o).

Operador lógico	Símbolo	Operación	Expresión lógica	Significado
Not (no)	~	Negación	Not P (no P)	Negación de P
And (y)	^	Conjunción	P and Q (P y Q)	Intersección de P y Q
Or (o)	V	Disyunción	P or Q (P o Q)	Unión de P y Q

**Tabla 3:** Elaboración propia**Asignación**

La operación de asignación es el modo de darle valores a una variable. La operación de asignación se representa con el símbolo u operador “=”. También se conoce como instrucción o sentencia de asignación cuando se refiere a un lenguaje de programación. Por ejemplo, la operación de asignación:

**A=5**

establece que a la variable A se asigne el valor 5.

La acción de asignar es destructiva, ya que el valor que tuviera la variable antes de la asignación se pierde y se reemplaza por el nuevo valor. Así en la secuencia de operaciones:

A = 25

A = 134

A = 5

Cuando éstas se ejecutan, el valor último que toma A será 5 (los valores 25 y 134 han desaparecido)

La computadora ejecuta la secuencia de asignación en dos pasos. En el primero de ellos, el valor de la expresión al lado derecho del operador se calcula, obteniéndose un valor de un tipo específico. En el segundo paso este valor se almacena en la variable cuyo nombre aparece a la izquierda del operador de asignación, sustituyendo al valor que tenía anteriormente.

X = Y + 2

el valor de la expresión Y + 2 se asigna a la variable X.

Ejemplos de asignación: ¿Cuál será el valor que tomará la variable C tras la ejecución de las siguientes instrucciones?

A = 12

B = A

C = B



Respuesta:

A contiene 12, B contiene 12 y C contiene 12.

NOTA: Antes de la ejecución de las instrucciones, el valor de A, B y C es indeterminado. Si se desea darles un valor inicial, habrá que hacerlo explícitamente, incluso cuando este sea 0. Es decir, habrá que definir e inicializar las instrucciones.

A = 0

B = 0

C = 0

¿Cuál es el valor de  $N = N + 5$ , si N tiene el valor actual de 2?

Se realiza el cálculo de la expresión  $N + 5$  y su resultado  $2 + 5 = 7$ , se asigna a la variable situada a la izquierda, N tomará un nuevo valor. Se debe pensar en la variable como en una posición de memoria, cuyo contenido puede variar mediante instrucciones de asignación.

#### 4. LECTURA DE DATOS (ENTRADA)

Esta instrucción lee datos de un dispositivo de entrada. ¿Cuál será el significado de las instrucciones siguientes?

Leer NUMERO, HORAS, TASA

Leer (también se utiliza Ingresar) del teclado los valores NUMERO, HORAS, TASA, archivándolos en la memoria; si los tres números que se teclean en respuesta a la instrucción son: 12, 32, 45, significaría que se han asignado a las variables esos valores y equivaldría a la ejecución de las instrucciones.

NUMERO = 12

HORAS = 32

TASA = 45

#### 5. ESCRITURA DE RESULTADOS (SALIDA)

Esta instrucción se escribe o imprime en un dispositivo de salida.

El significado de la ejecución de las siguientes instrucciones:

A = 100

B = 200

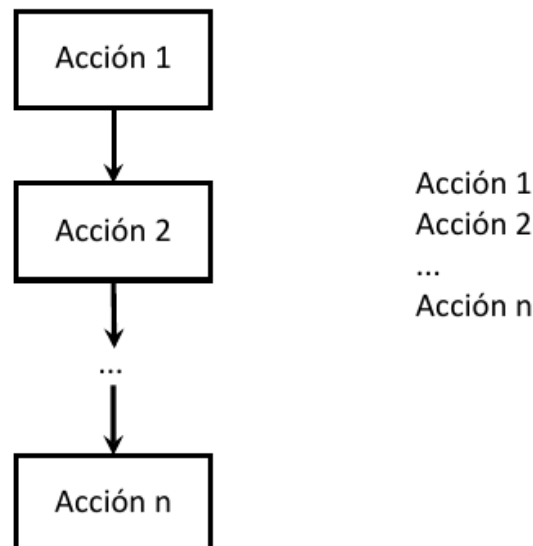
C = 300

Escribir A, B, C

puede interpretarse como: se visualizarán en la pantalla o imprimirán en la impresora los valores 100, 200, 300 que contienen las variables A, B, C.

**MODULO 3. ESTRUCTURAS SECUENCIALES**

Son aquellas en la que una acción sigue a otra en secuencia. Las tareas se suceden de tal modo que al finalizar una acción se inicia inmediatamente la siguiente. Gráficamente se representa de la siguiente manera:



**Gráfico 4:** Elaboración propia

**Problema modelo**

Problema 1.1: Un profesor necesita obtener el promedio de notas de un alumno. Se conoce que el alumno ha rendido tres exámenes parciales. Se pide generar la siguiente *Salida Impresa*:

- El resultado del promedio del alumno.

Para ello Ud. dispone de las siguientes *Entradas*:

- Nota1 (n1): representa la nota del primer parcial.
- Nota2 (n2): representa la nota del segundo parcial.
- Nota2 (n3): representa la nota del tercer parcial.

## 3.1 Diagrama de flujo

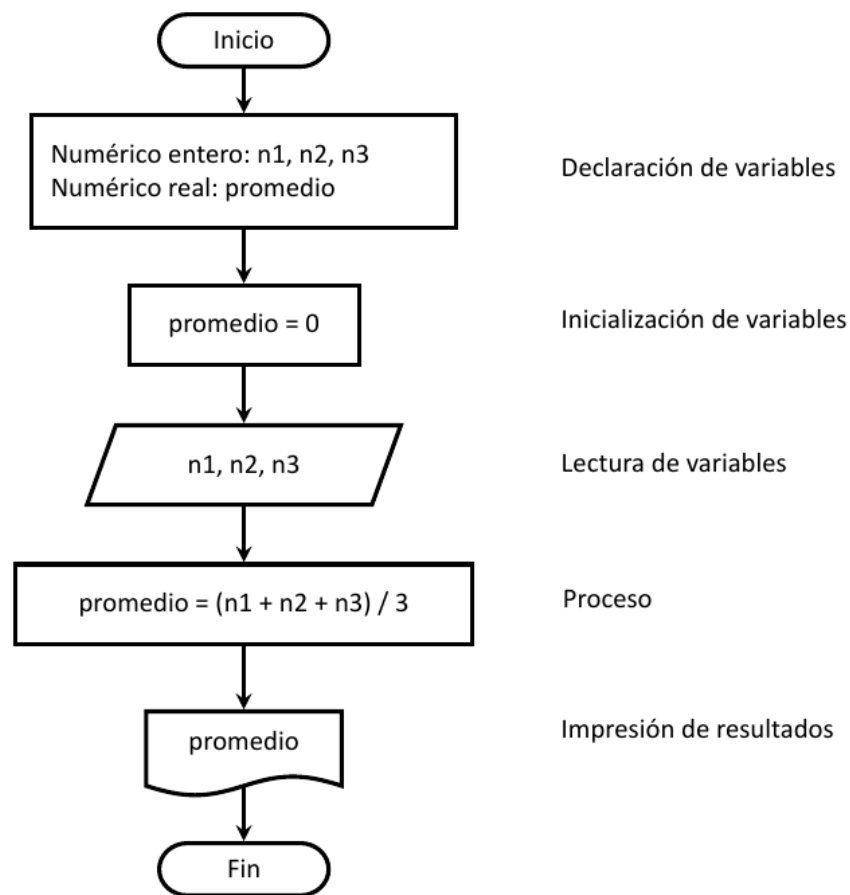


Gráfico 5: Elaboración propia

## Prueba de Escritorio

Variables			
n1	n2	n3	promedio
0	0	0	0
4	4	6	4,6666667
6	10	4	6,6666667
3	4	6	4,3333333
6	5	7	6

Gráfico 6: Elaboración propia

**MODULO 4: ESTRUCTURAS CONDICIONALES**

Son aquellas que dirigen el flujo de ejecución según el resultado de evaluación de expresiones lógicas. Se utilizan para tomar decisiones lógicas; de ahí que se suelen denominarse también estructuras de decisión o alternativas.

En la estructura selectiva se evalúa una condición y en función del resultado de la misma se realiza una opción u otra. Las estructuras selectivas o alternativas pueden ser:

- Simples
- Dobles
- Múltiples

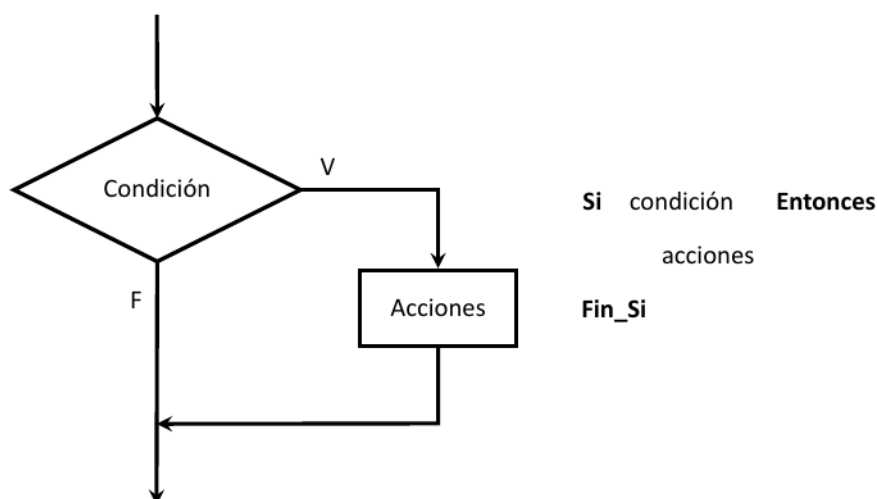
**1. ALTERNATIVA SIMPLE**

La estructura alternativa simple “**si-entonces**”, ejecuta una determinada acción cuando se cumple una cierta condición.

La selección si-entonces evalúa la condición:

- Si la condición es verdadera, entonces ejecuta la acción (la cual puede ser simple o contar de varias acciones)
- Si la condición es falsa, entonces no hace nada.

La representación gráfica de la estructura condicional simple se ve a continuación.



**Gráfico 7:** Elaboración propia

Debe observarse que las palabras del pseudocódigo Si y Fin\_Si se alinean verticalmente indentando (sangrando) la acción o bloque de acciones.

## 2. ALTERNATIVA DOBLE

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición. El comportamiento de la alternativa doble es el de evaluar la condición y a continuación:

- Si la condición es verdadera, se ejecuta la acción 1.
- Si la condición es falsa, se ejecuta la acción 2

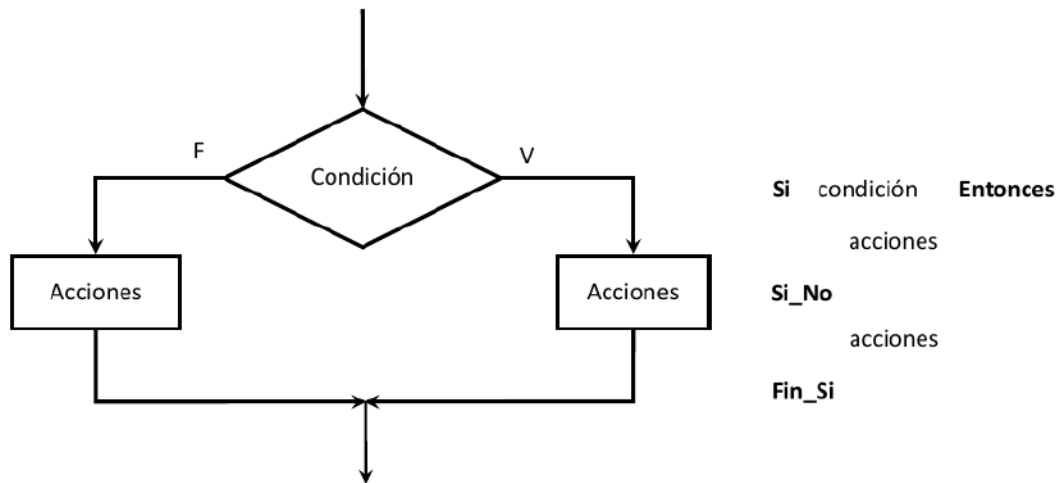


Gráfico 8: Elaboración propia

## 3. ALTERNATIVA MÚLTIPLE

Con frecuencia, es necesario que existan más de dos elecciones posibles. Por ejemplo, en la resolución de la ecuación de segundo grado existen tres posibles alternativas o caminos a seguir, según que el discriminante sea negativo, nulo o positivo.

Este problema se podría resolver por estructuras alternativas simples o dobles, anidadas o en cascada; sin embargo, con este método, si el número de alternativas es grande puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad.

La estructura de decisión múltiple evaluará una expresión que podrá tomar  $n$  valores distintos: 1, 2, 3, ...,  $n$ . Según que elija uno de estos valores en la condición, se realizará una de las  $n$  acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los  $n$  posibles.

Gráficamente:

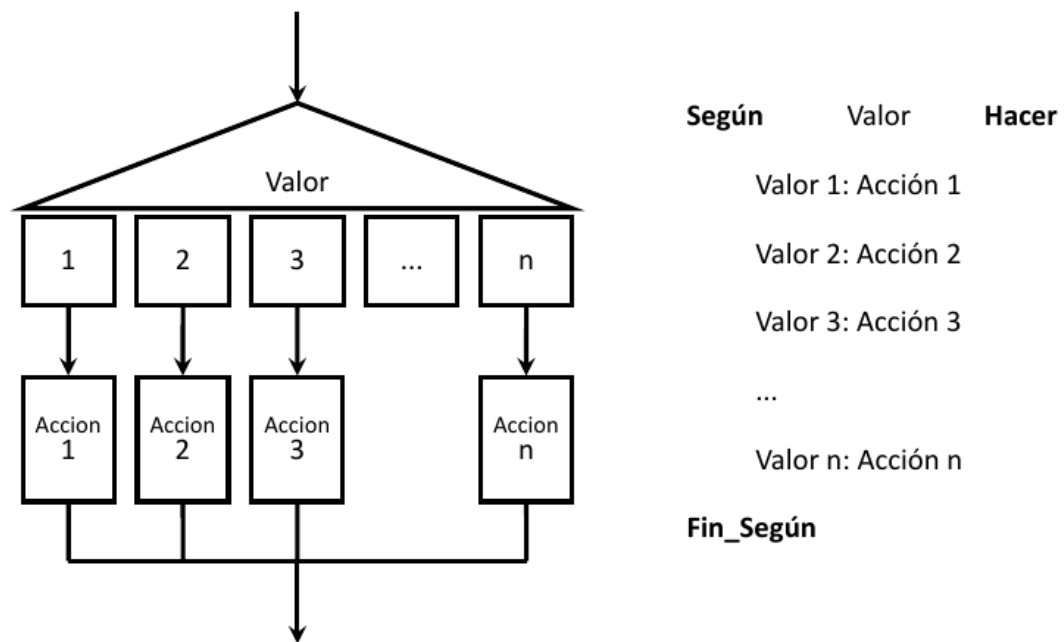


Gráfico 9: Elaboración propia

La estructura de decisión múltiple en pseudocódigo se puede representar de diversas formas, pudiendo ser las acciones, simples o compuestas.

#### Problema Modelo

Problema 1.2: Un docente necesita obtener información relacionada con el promedio de un alumno. Se pide generar las siguientes salidas:

- El promedio del alumno.
- La condición académica del alumno. La condición será alumno regular si el promedio es mayor o igual a 4, y reprobado si su promedio final es menor a 4.

Para ello Ud. dispone de las siguientes entradas:

- Nota1 (n1): representa la primera nota de examen del alumno.
- Nota2 (n2): representa la segunda nota de examen del alumno.
- Nota2 (n3): representa la tercera nota de examen del alumno.

## Diagrama de Flujo

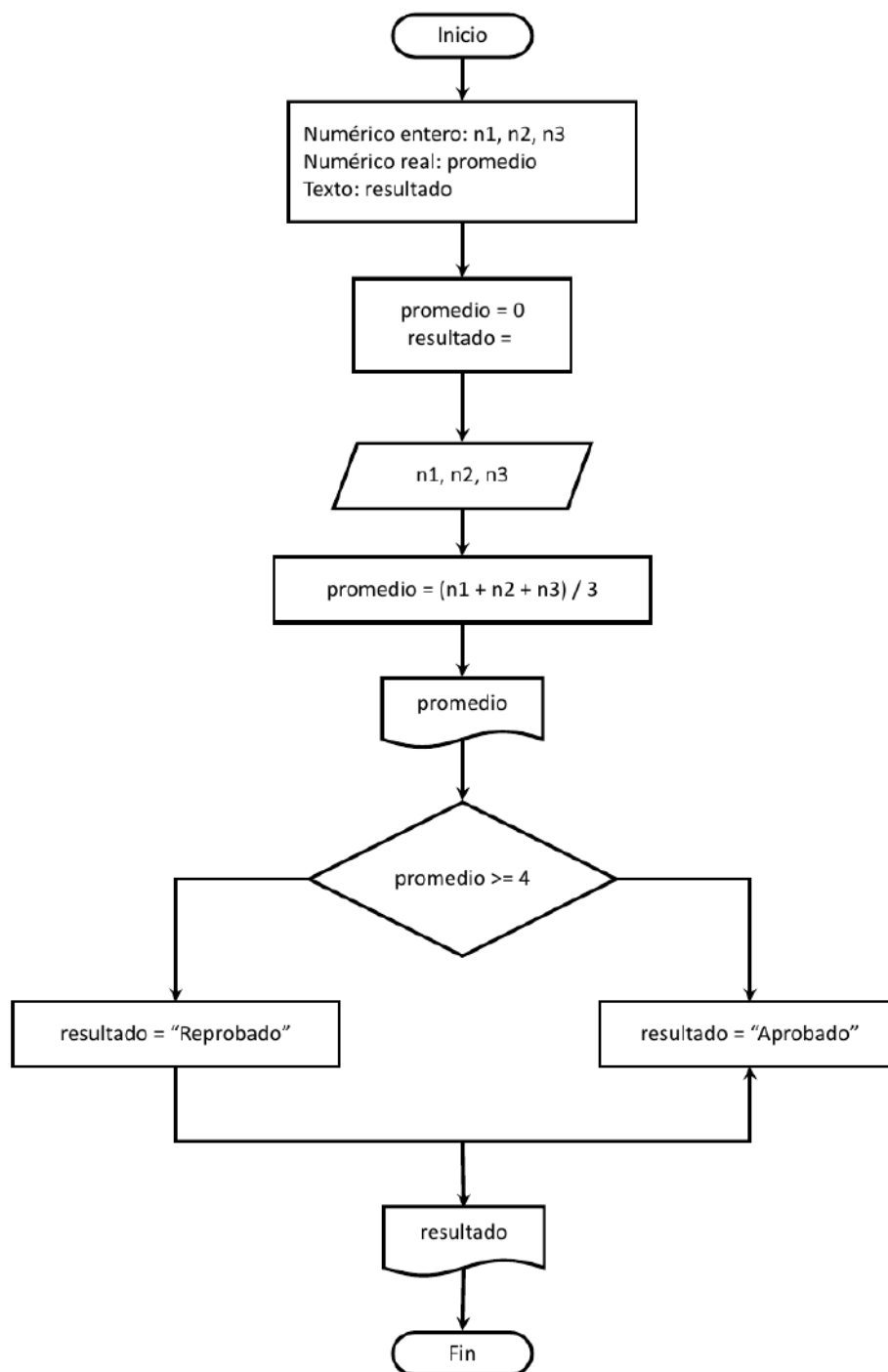


Gráfico 10: Elaboración propia

## Prueba de Escritorio

Variables				
n1	n2	n3	promedio	resultado
0	0	0	0	
4	4	6	4,6666667	Regular
6	10	4	6,6666667	Regular
3	4	6	4,3333333	Regular
2	4	3	3	Reprobado

Gráfico 11: Elaboración propia



**MODULO 5: ESTRUCTURAS REPETITIVAS**

Las estructuras repetitivas permiten ejecutar un conjunto de sentencias repetidamente una cierta cantidad de veces o hasta que se cumpla una determinada condición. El conjunto de sentencias se denomina bucle y cada repetición del cuerpo del bucle se denomina iteración.

Un bucle o lazo (loop) es un segmento de un algoritmo o programa, cuyas instrucciones se repiten un número determinado de veces mientras se cumple una determinada condición. Se debe establecer un mecanismo para determinar las tareas repetitivas. Este mecanismo es una condición que puede ser verdadera o falsa y que se comprueba una vez a cada paso o iteración del bucle.

Todo bucle consta de tres partes:

1. Decisión.
2. Cuerpo del bucle.
3. Salida del bucle.

Por ejemplo: Si se desea sumar una lista de números escritos desde teclado. El medio conocido hasta ahora es leer los números y añadir sus valores a una variable SUMA que contenga las sucesivas sumas parciales. La variable SUMA se hace igual a cero y a continuación se incrementa en el valor del número cada vez que uno de ellos se lea.

El algoritmo que resuelve este problema es:

```
Inicio  
suma = 0  
leer numero  
suma = suma + numero  
leer numero  
suma = suma + numero  
...  
Fin
```

Así sucesivamente para cada número de la lista. En otras palabras, el algoritmo repite muchas veces las acciones. Las dos principales preguntas a realizarse en el diseño de un bucle son: ¿Qué contiene el bucle? y ¿Cuántas veces se debe repetir?

Cuando se utiliza un bucle para sumar una lista de números, se necesita saber cuántos números se han de sumar. Para ello necesitaremos conocer algún medio para detener el bucle. Existen dos procedimientos para contar el número de iteraciones, usar una variable que se inicializa a la cantidad de números que se desea y a continuación se decrementa en uno (-1) cada vez que el bucle se repite, o bien inicializar la variable en cero o en uno, e ir incrementando en uno (+1) a cada iteración hasta llegar al número deseado.

Para detener la ejecución de los bucles se utiliza una condición de parada. Los tres casos generales de estructuras repetitivas dependen de la situación y modo de la condición. La condición se evalúa tan pronto se encuentra en el algoritmo y su resultado producirá los tres tipos de estructuras citadas.

- La condición de salida del bucle se realiza al principio del bucle (estructura mientras)
- La condición de salida se origina al Final del bucle; el bucle se ejecuta mientras o hasta que se verifique una cierta condición.
- La condición de salida se realiza con un contador que cuenta el número de iteraciones.

## 1. ESTRUCTURA MIENTRAS

La estructura repetitiva mientras es aquella en la que el bucle se repite mientras se cumple una determinada condición. La representación gráfica es:

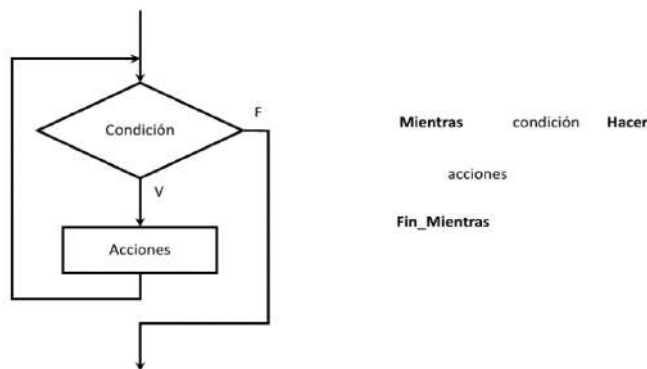
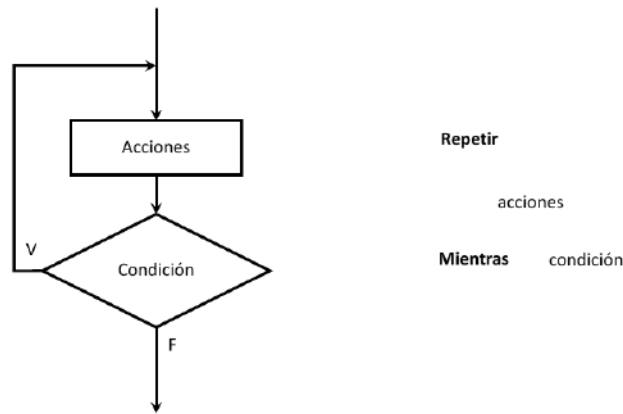


Gráfico 12: Elaboración propia

Cuando se ejecuta la instrucción mientras, la primera cosa que sucede es que se evalúa la condición (una expresión booleana). Si se evalúa falsa, ninguna acción se toma y el programa prosigue en la siguiente instrucción. Si la expresión booleana es verdadera, entonces se ejecuta el cuerpo del bucle, después de lo cual se evalúa de nuevo la expresión booleana. Este proceso se repite una y otra vez mientras la expresión (condición) sea verdadera.

## 2. ESTRUCTURA REPETIR O HACER MIENTRAS

Existen muchas situaciones en las que se desea que un bucle se ejecute al menos una vez antes de comprobar la condición de repetición. En la estructura mientras si el valor de la expresión booleana es inicialmente falsa, el cuerpo del bucle no se ejecutará; Por ello se necesitan otros tipos de estructuras repetitivas. La estructura repetir o hacer mientras se ejecuta mientras que se cumpla una condición determinada que se comprueba al final del bucle.



**Gráfico 13:** Elaboración propia

El bucle repetir mientras se repite mientras el valor de la expresión booleana de la condición sea verdadera.

Con una estructura repetir, el cuerpo del bucle se ejecuta siempre al menos una vez. Cuando una estructura repetir se ejecuta, la primera cosa que sucede es la ejecución del bucle y a continuación se evalúa la expresión booleana resultante de la condición. Si se evalúa como verdadera, el cuerpo del bucle se repite y la expresión booleana se evalúa una vez más. Después de cada iteración del cuerpo del bucle, la expresión booleana se evalúa; si es falsa, el bucle termina y el programa continúa en la siguiente instrucción. Un uso típico de este tipo de estructura es la validar datos de entrada, tal como se muestra en los casos modelos desarrollados más adelante.

## a. Diferencias de las Estructuras Mientras y Repetir

Estructura Mientras	Estructura Hacer o Repetir Mientras
Es un ciclo de 0 a N, porque si la condición inicial no se cumple no se entra al ciclo, por lo tanto existe la posibilidad de que las acciones del ciclo nunca sean ejecutadas.	Es un ciclo de 1 a N, porque siempre se ejecuta al menos una vez las instrucciones del ciclo, si la primera vez la condición inicial es falsa, al evaluar la condición al final, se termina el ciclo pero ya se ejecutaron por lo menos una vez las acciones.

Tabla 4: Elaboración propia

## 3. ESTRUCTURA DESDE / PARA

En muchas ocasiones se conoce de antemano el número de veces que desean ejecutar las acciones de un bucle. En estos casos, en que el número de iteraciones es fija, se debe usar la estructura desde / para.

La estructura **desde** ejecuta las acciones del cuerpo del bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.

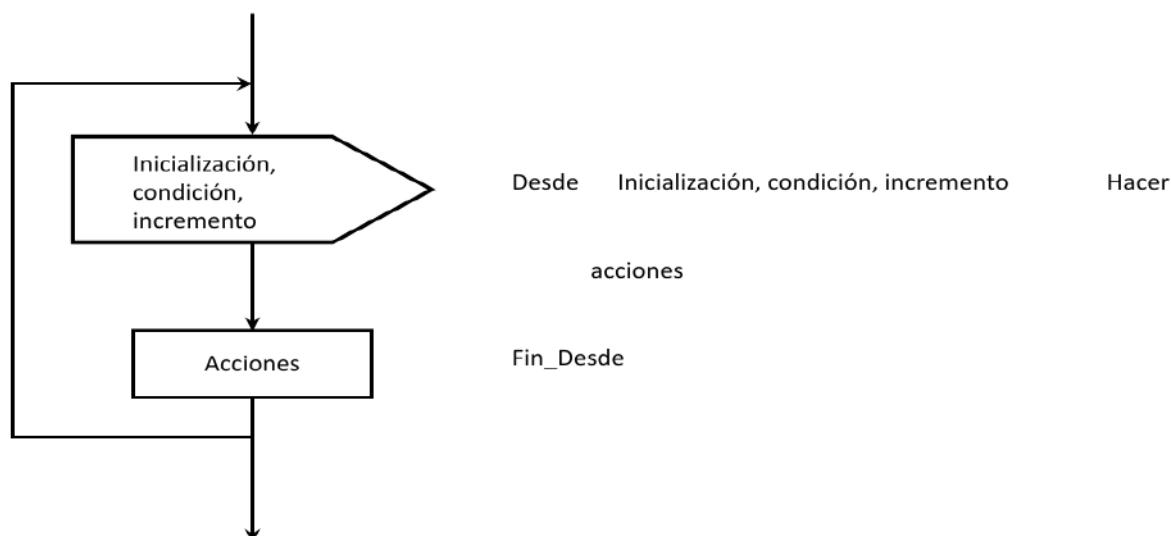


Gráfico 15: Elaboración propia

La estructura desde comienza con un valor inicial de la variable índice y las acciones especificadas se ejecutan a menos que el valor inicial sea mayor que el valor final. La variable índice se incrementa en uno y si este nuevo valor no excede al final, se ejecutan de nuevo las acciones. Por consiguiente, las acciones específicas en el bucle se ejecutan para cada valor de la variable índice desde el valor inicial hasta el valor final con el incremento de uno en uno.

El incremento de la variable índice siempre es 1 si no se indica expresamente lo contrario. Es posible que el incremento sea distinto de uno, positivo o negativo.

La variable índice o de control normalmente será de tipo entero y es normal emplear como nombre las letras: I, J, K.

El formato de la estructura desde varía si se desea un incremento distinto a 1, bien positivo, bien negativo (decremento). Si el valor inicial de la variable índice es menor que el valor final, los incrementos deben ser positivos, ya que en caso contrario la secuencia de acciones no se ejecutaría. De igual modo si el valor inicial es mayor que el valor final, el incremento debe de ser en este caso negativo, es decir, decremento. Al incremento se le suele denominar también paso.

#### Problema modelo estructura repetitiva mientras

Problema 1.3: Un docente necesita obtener información relacionada con los promedios de alumnos de un curso. Se pide generar las siguientes salidas impresas:

- El promedio del alumno.
- El promedio general del curso.
- La cantidad de alumnos regulares y libres. La condición será alumno regular si el promedio es mayor o igual a 4, y reprobado si su promedio final es menor a 4.
- Para ello Ud. dispone de las siguientes entradas:
- Legajo (leg): representa el legajo del alumno.
- Nota1 (n1): representa la primera nota de examen del alumno.
- Nota2 (n2): representa la segunda nota de examen del alumno.
- Nota2 (n3): representa la tercera nota de examen del alumno.
- Deberán ingresarse alumnos hasta que el usuario ingrese un legajo menor o igual a 0.

## Diagrama de flujo

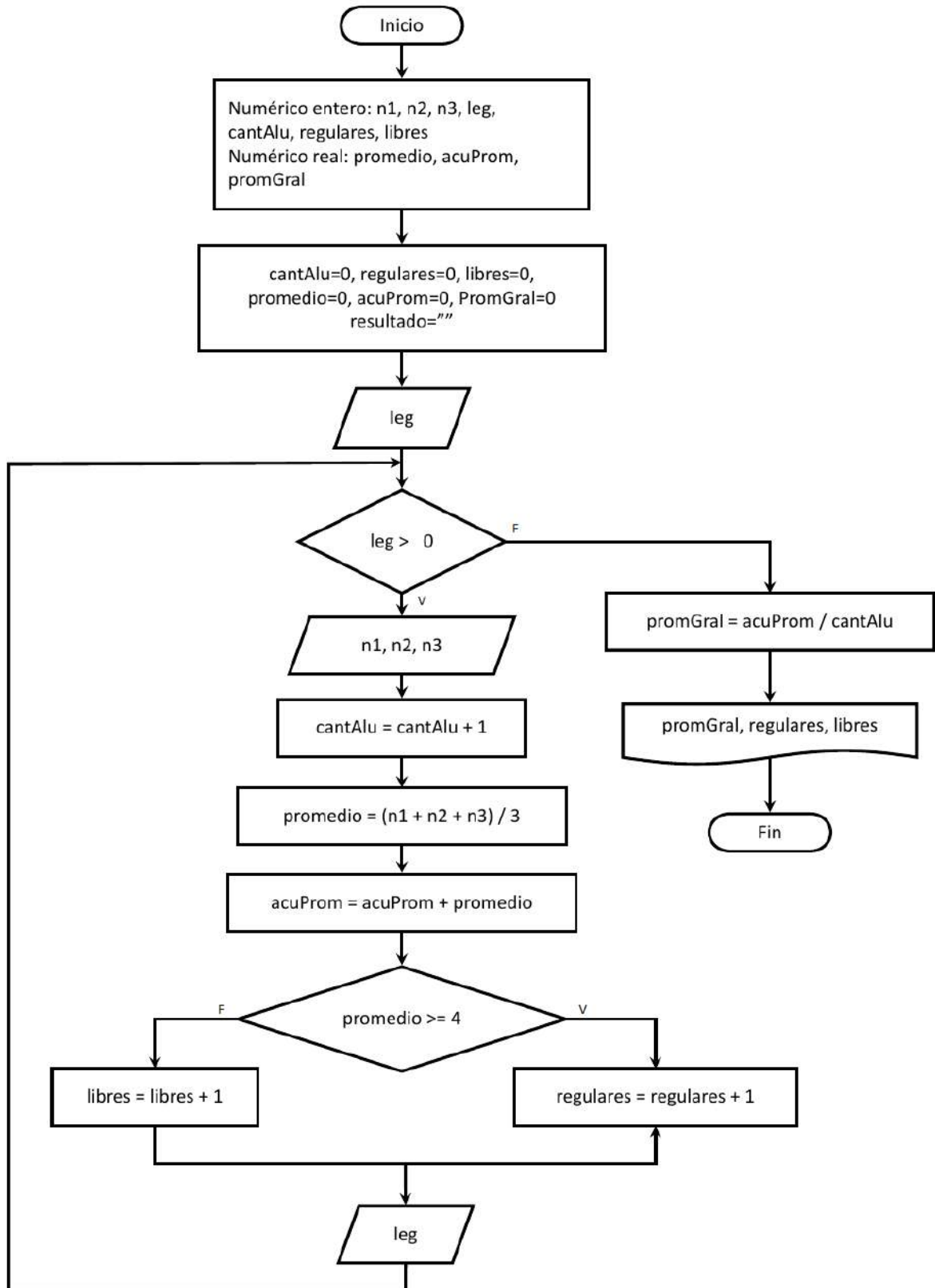


Gráfico 16: Elaboración propia

## Prueba de escritorio

Legajo	Variables								
	n1	n2	n3	promedio	acuProm	cantAlu	promGral	regulares	libres
0	0	0	0	0	0	0	0	0	0
64567	4	4	6	4,6666667	4,67	1	0	1	0
65456	10	10	10	10,00	14,67	2	0	2	0
64345	3	2	2	2,33	17,00	3	0	2	1
63454	2	2	3	2,3333333	19,34	4	0	2	2
65677	9	4	6	6,3333333	25,67	5	0	3	2
0	9	4	6	6,3333333	32,00	5	6,400667	3	2

Tabla 5: Elaboración propia

## Problema modelo estructura repetir mientras

Problema 1.4: Un docente necesita obtener información relacionada con los promedios de los alumnos de un curso.

Se pide generar las siguientes salidas impresas:

- El promedio de los alumnos.
- El resultado de validar que el promedio ingresado sea válido. Es decir que se ingrese un valor entre 1 y 10.
- Cuántos alumnos han obtenido la promoción mayor o igual a 7.  
Para ello Ud. dispone de las siguientes entradas:
- Legajo (leg): representa el legajo del alumno.
- Promedio (prom): representa el promedio de notas del alumno al finalizar el curso.

Considerar que deberá ingresar alumnos, hasta que el usuario ingrese un legajo igual a 0.

## Diagrama de flujo

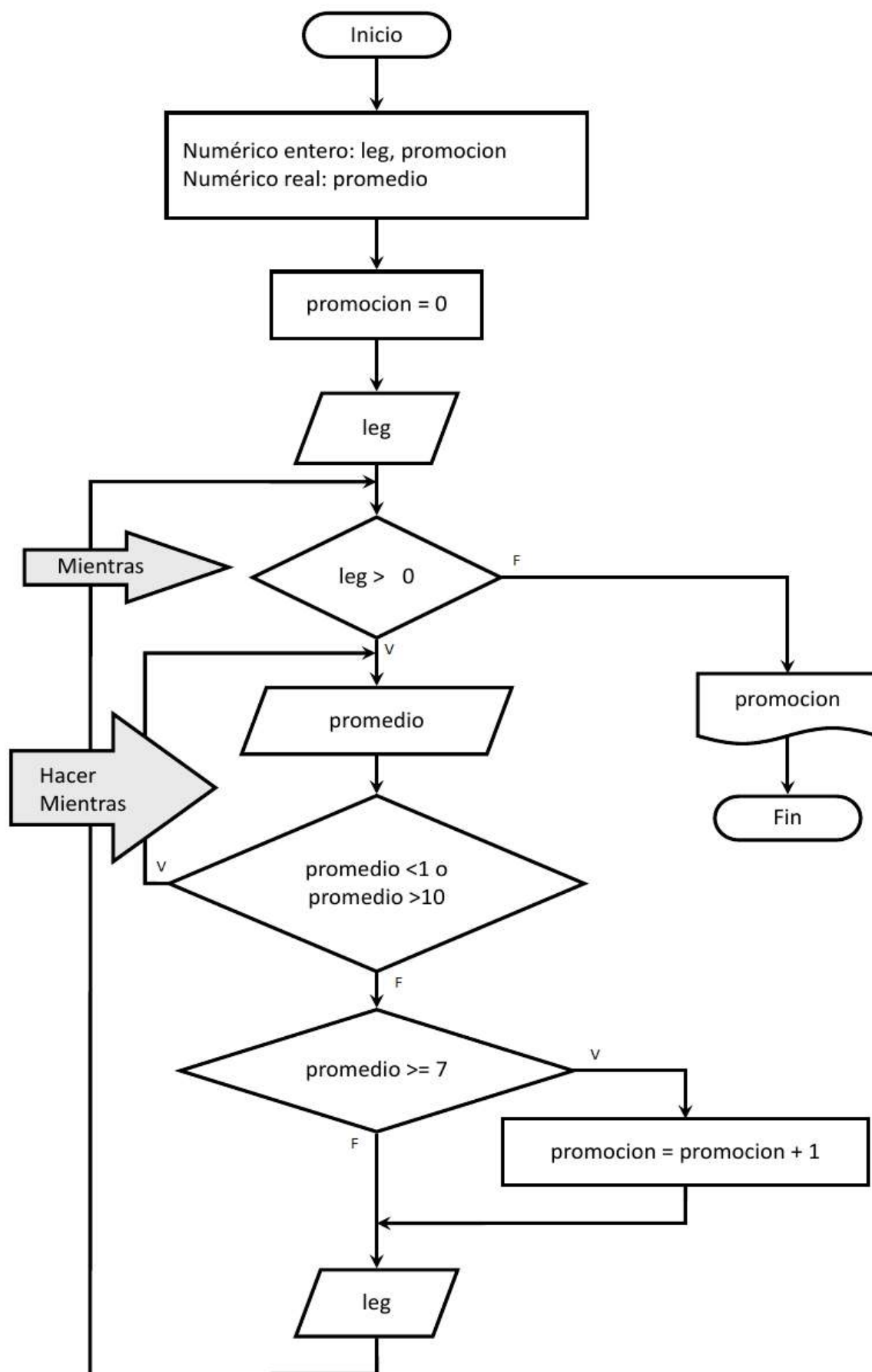


Gráfico 17: Elaboración propia



## Prueba de escritorio

Variables		
Legajo	Promedio	Promocion
0	0	0
64567	4,5	0
65456	10	1
64345	11	1
63454	12	1
65677	8	2
0	8	2

**Gráfico 18:** Elaboración propia

## Problema modelo estructura desde / para

Problema 1.5: Un docente necesita obtener información relacionada con el promedio de los alumnos de un curso que tiene 50 alumnos. Se pide generar las siguientes salidas impresas:

- El promedio de los alumnos.
- La cantidad de alumnos cuya nota promedio está comprendida entre 4 y 10.

Para ello Ud. dispone de la siguientes Entradas:

- Legajo (leg): representa el legajo del alumno.
- Nota1 (n1): representa la primer nota de examen del alumno.
- Nota2 (n2): representa la segunda nota de examen del alumno.
- Nota2 (n3): representa la tercera nota de examen del alumno.

## Diagrama de flujo

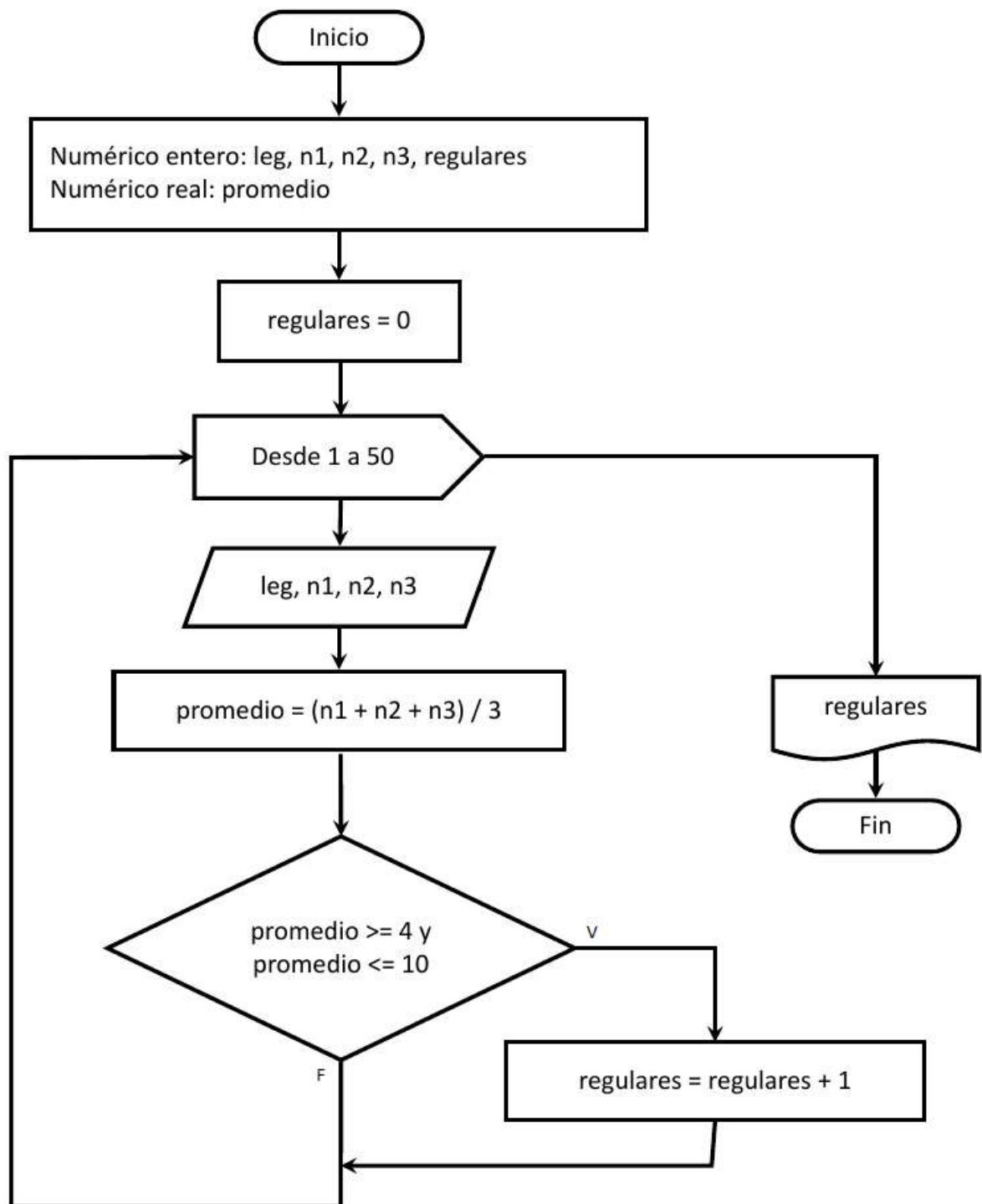


Gráfico 19: Elaboración propia

## Prueba de escritorio

legajo	Variables				
	n1	n2	n3	promedio	regulares
0	0	0	0	0	0
64567	4	4	6	4,67	1
65456	6	10	4	6,67	2
64345	3	4	6	4,33	3
63454	2	4	3	3,00	3

Gráfico 20: Elaboración propia

## 6. CASOS DE ESTUDIO

Es muy frecuente cuando se comienza a resolver problemas de estructuras del tipo repetitiva, que se necesita procesar una serie o conjunto significativo de elementos como por ejemplo: alumnos, docentes, operarios etc.

En esta naturaleza de problemas a resolver en oportunidades surge la necesidad de buscar precisamente en esa serie un mayor o menor. Por ejemplo, si el enunciado plantea la necesidad de manejar operarios, puede ser que una de las salidas solicitadas sea: mostrar el legajo del operario que produjo una mayor cantidad de piezas o el que produjo menos cantidad de piezas. A continuación, se explicará una de las estrategias de resolución más eficientes para poder obtener el mayor o menor de una serie.

## Problema Modelo

Problema 1.6: Un profesor necesita obtener información relacionada con los promedios de notas de los alumnos.

Se pide generar la siguiente salida impresa:

- El legajo del alumno que obtuvo el mayor promedio.  
Para ello se dispone de las siguientes entradas:
- Legajo (leg): representa el legajo del alumno.
- Promedio (prom): representa el promedio final del alumno.

Considerar que la carga del alumno finalizará cuando se ingrese un legajo igual a cero.

### Estrategia propuesta

La salida principal solicitada por el problema es mostrar el legajo del alumno que obtuvo el mayor promedio. Para dar respuesta a esta salida, una de las alternativas es seguir los siguientes pasos:

1. Definimos una variable mayor y la inicializaremos con el primer valor ingresado en la serie. Para el caso de nuestro problema será el primer promedio ingresado. La idea general es tomar el primer promedio de la serie para considerarlo como factor de comparación con los demás promedios ingresados y obtener el mayor.
2. En el ciclo mientras procesamos los alumnos, debemos asegurarnos de almacenar en mayor el primer promedio ingresado. ¿Pero cómo hacemos para que dentro del ciclo solo una única vez pase el flujo de instrucciones, con la finalidad de asignar en la variable mayor el primer promedio ingresado? La respuesta es hacer uso de una variable, para nuestro caso llamada var, inicializada en 1, y dentro del ciclo preguntamos si esa variable es igual a "1". La primera vez que se entra al ciclo la respuesta a esta pregunta será verdadera, en ese caso asignamos a la variable mayor el primer promedio ingresado. Y dentro de la rama verdadera de esta condicional asignaremos a var en "0". La idea de asignar el valor "0" a la variable var es cuando vuelva el flujo de instrucciones a entrar al ciclo, la condición verdadera de la condición no se vuelva a ejecutar. Y hemos cumplido con nuestro propósito, es decir dentro de nuestro ciclo sólo una única vez asignamos a var con el primer promedio ingresado
3. El tercer paso es comparar los demás promedios ingresados con la variable var que almacena nuestro primer promedio ingresado y lo hemos tomado como punto de comparación para obtener el valor del mayor promedio.
4. Si el promedio leído es mayor var se almacena el promedio leído en la variable mayor. Así será sucesivamente hasta que se termine de procesar los alumnos.

Sin embargo debe recordarse que se requiere el legajo de aquel alumno que obtuvo el mayor promedio, por lo tanto debe almacenarse dicho legajo en alguna variable. Para ello utilizamos la variable legmay.

## Diagrama de flujo

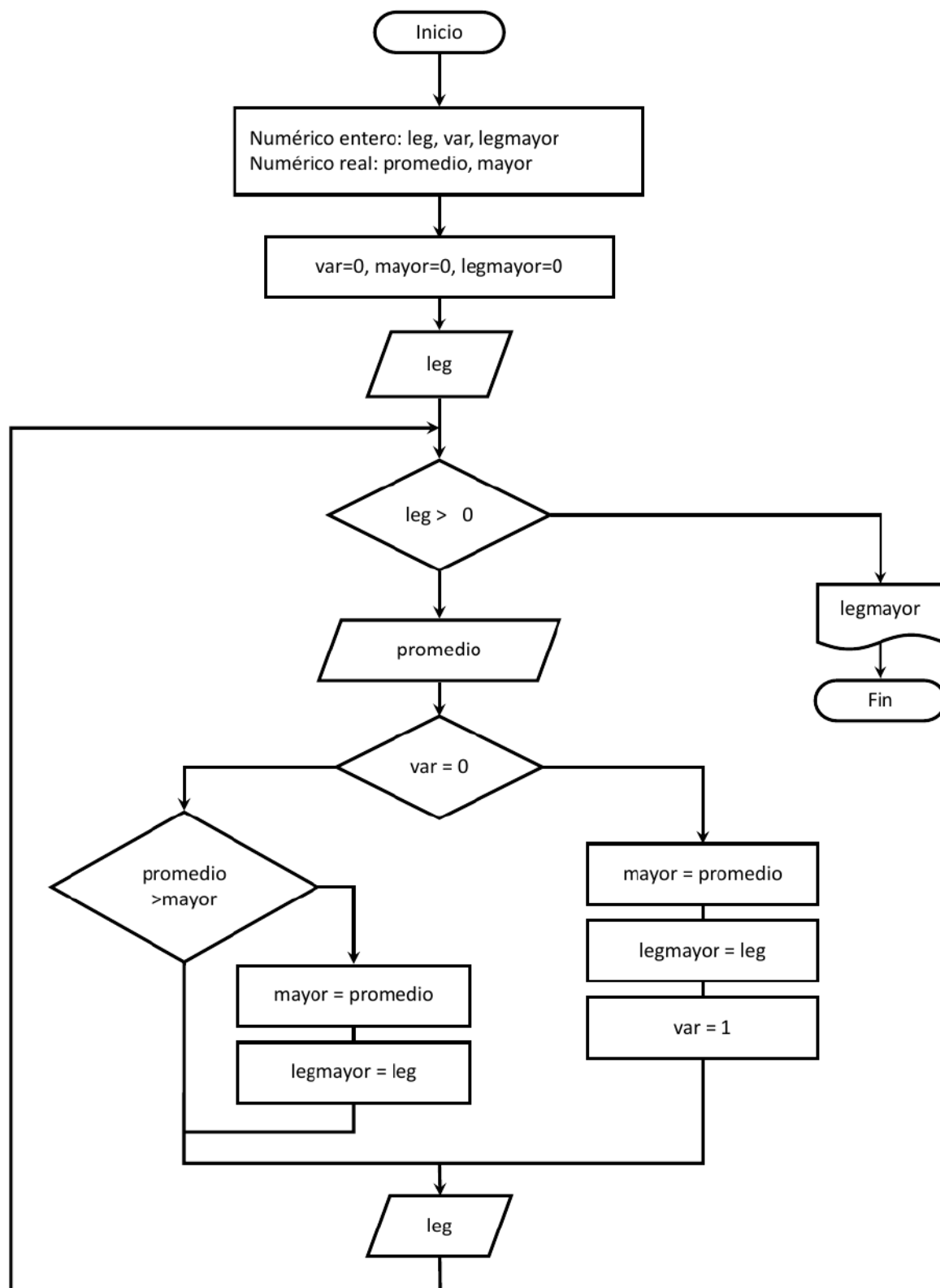


Gráfico 21: Elaboración propia

## Prueba de escritorio

Variables				
leg	promedio	var	legmay	mayor
0	0	0	0	0
64567	4,5	1	64567	4,5
65456	10	1	65456	10
64345	8,75	1	65456	10
63454	6	1	65456	10
0	6	1	65456	10

**Gráfico 22:** Elaboración propia

## BIBLIOGRAFÍA

Bishop, P. (1992). Fundamentos de Informática. Anaya.

Brookshear Computer Science. (1994). An Overview. Benjamin/Cummings.

Joyanes, L. (1990). Problemas de Metodología de la Programación. McGraw Hill.

Joyanes, L. (1993). Fundamentos de Programación: Algoritmos y Estructura de Datos. McGraw Hill.

De Miguel, P. (1994). Fundamentos de los Computadores. Paraninfo.

Norton, P. (1995). Introducción a la Computación. McGraw Hill.

Prieto, A. Lloris J. & Torres, C. (1989). Introducción a la Informática. McGraw Hill.

Tucker, A. Bradley, W, Cupper, R Garnick, D (1994) Fundamentos de Informática: Lógica, resolución de problemas, programas y computadoras. McGraw Hill.

### Atribución-NoComercial-SinDerivadas



Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterado su contenido, ni se comercialice. Referenciarlo de la siguiente manera:

Universidad Tecnológica Nacional Facultad Regional Córdoba. Material para la Tecnicatura Universitaria en Programación modalidad virtual Córdoba, Argentina.