4. CONSTRUCTORES BÁSICOS

4.1 LOS DATOS EN PROGRAMACIÓN

4.1.1 Datos

Un **conjunto** es una colección *'bien definida'* de objetos. Se dice *'bien definida'* si se sabe de manera exacta que elementos están en la colección y que elementos no están. Existen dos maneras de definir un conjunto: por extensión y por comprensión.

Un conjunto es definido por **extensión** cuando se presentan todo sus elementos entre llaves **{}**.

Ejemplos.

A = {a, b, c}
B = {0, 1, 2, 3, 4}

Un conjunto es definido por comprensión cuando sus elementos no se mencionan uno a uno, sino que se describen enunciando una propiedad que todos ellos cumplen.

Ejemplos.

C = {x | x es un número primo}
D = {x | x es una vocal}

A cada conjunto se le puede asignar uno o varios nombres; dichos nombres constituyen el **tipo** de los elementos del conjunto.

De manera informal, un **dato** es una pieza de información simple como un número, un código, un hecho o una edad. De manera formal, un **dato** es un elemento concreto de algún conjunto. El nombre del conjunto al que pertenece el dato constituye el tipo del mismo.

Los tipos de datos más utilizados en programación son:

		EJEMPLOS
Entero:	El nombre asignado al conjunto de números enteros.	123, -1, 0, 2
Real:	El nombre asignado al conjunto de números reales.	0.123, 3.1415, -2.0
Caracter:	El nombre asignado al conjunto de caracteres.	'a', '*', '+', ';'
Booleano:	El nombre asignado al conjunto de valores de verdad.	falso, verdadero
Cadena:	El nombre asignado al conjunto de cadenas de caracteres.	"Hola", "AbC123"

Como se puede advertir en los ejemplos anteriores, cuando se trabaja en

programación es necesario:

- *Distinguir los números enteros de los números reales*. Por ejemplo el **2** (sin punto decimal), indica que el elemento es del conjunto de los enteros, mientras que el **2.0** (con punto decimal), indica que es un elemento del conjunto de los reales.
- *Distinguir los símbolos y palabras que forman parte del sistema de representación o codificación del algoritmo, de los usados para representar un dato concreto de los tipos caracter y cadena de caracteres respectivamente*. En este curso, los datos de tipo caracter son representados entre comillas simples (‘), mientras que los datos de tipo cadena de caracteres son representados entre comillas dobles (“).

4.1.2 Variables

Una **variable** es un símbolo que permite referenciar (señalar o demarcar) un espacio en memoria en el que se puede almacenar un dato. Toda variable posee tres características: nombre, tipo y estado.

4.1.2.1 Nombre o identificador.

El nombre de una variable es una secuencia de caracteres alfanuméricos (letras y dígitos) y caracteres de subrayado, la cual siempre empieza por una letra. En algunos lenguajes de programación (C++, C, Java), se hace distinción entre mayúsculas y minúsculas mientras que en otros no (Pascal, Basic). Por ejemplo, las variables *VelMax* y *velmax* son diferentes en *C* mientras que son la misma variable en *Pascal*.

Una buena técnica de programación es asignarle el nombre a una variable de tal manera que indique por un lado el papel que desempeña dicha variable en el algoritmo y por otro los posibles valores que almacena.

Ejemplos de nombre de variable.

- velocidad
- x
- valor1
- exponente
- valMaximo

4.1.2.2 Tipo.

El tipo de una variable es el mismo tipo de los datos que se pueden almacenar en el espacio de memoria que la variable referencia. Por ejemplo, una variable de tipo entero, es decir, que se ha declarado[1] como entera, sólo se puede usar para referenciar datos de tipo entero. La cantidad de memoria, es decir, el espacio en

memoria en bits, asignada para almacenar un dato de un tipo depende del sistema operativo y del compilador del lenguaje.

En este curso una variable es declarada de la siguiente manera:

<nombre> : **<tipo>**

Donde, **<nombre>** es el nombre que se le dará a la variable y **<tipo>** es el tipo de datos que la variable puede referenciar.

Ejemplos de declaración de variables.

- valMaximo : real
- contador : entero
- nombre : cadena
- letra : caracter
- bandera : booleano

4.1.2.3 Estado o valor

El estado o valor de una variable es el dato almacenado en el espacio que referencia la variable. El valor de una variable puede cambiar en el tiempo, es decir, conforme el algoritmo se va ejecutando, pero solamente puede tomar valores que pertenezcan al tipo de datos declarado para la variable.

4.1.3 Literales

Un **literal** es una secuencia de caracteres que representa un valor concreto. Los literales son clasificados de acuerdo al tipo de datos que representan. Las siguientes secciones describen la forma de los literales usada en este libro (esta representación es específica para cada lenguaje de programación).

4.1.3.1 Literales enteros

Un literal entero es una secuencia de dígitos (0,1,..,9), posiblemente precedida por el guión (-), que permite representar un dato de tipo entero.

Ejemplos de literales enteros.

- 12345
- 4768
- –138
- 2609
- 10

4.1.3.2 Literales reales

Un literal real es una secuencia de dígitos, posiblemente precedida por el guión, y en la cual debe aparecer el punto decimal (.) que permite representar un dato de tipo real. El punto decimal separa la parte entera de la parte decimal del número; si no existe punto decimal el literal es considerado como un literal entero y generalmente, si después del punto no hay dígitos, el literal se considera incorrecto[2]. Los siguientes son ejemplos de literales reales:

Ejemplos de literales reales.

- 3465.98
- 29.0
- -123.78
- 23.7e-4 (en notación científica 23.7e-4 equivale a $23.7 * 10^{-4)}$
- –3.9876

4.1.3.3 Literales de caracteres

Un literal de tipo carácter es una símbolo delimitado por comillas simples (‘) que permite representar al carácter ubicado entre las comillas simples.

Ejemplos de literales de carácter.

- 'a’ representa el caracter **a**
- ‘ ‘ representa el caracter espacio en blanco
- '3’ representa el caracter dígito **3**
- '$’ representa el caracter de pesos.
- '?’ representa el caracter interrogación de cierre.

4.1.3.4 Literales Cadenas

Un literal de cadena es una secuencia de símbolos, delimitada por comillas dobles, que sirve para representar la cadena de caracteres delimitada por las comillas dobles.

Ejemplos de literales de caracter.

- “Pepito va al colegio” , representa la cadena ***Pepito va al colegio***.
- “El área de la circunferencia (2*pi*r) es : ” , representa la cadena ***El área de la circunferencia (2*pi*r) es :***.

4.1.3.5 Literales booleanos

Son las palabras: falso y verdadero, las cuales son usadas para representar los dos valores de verdad.

4.1.4 Constantes

Una constante es un símbolo que permite referenciar un espacio en memoria en el que hay un dato almacenado que no se puede cambiar. En términos generales, una constante es una variable para la cual no se puede modificar su estado durante la ejecución del programa.

Las constantes se declaran de la siguiente manera:
<nombre> = **<literal>**
Donde, **<nombre>** es el nombre de la constante y <literal> es el literal que representa el valor de la constante.

Una buena técnica de programación es usar letras mayúsculas para los nombres de las constantes.

Ejemplos de constantes.

- PI = 3.1415926
- VELOCIDAD_LUZ = 300000000.0
- SALUDO_BASICO = “Hola, Buenos días”
- TAMANO_MAXIMO = 1000
- ESPACIO = ‘ ’

4.1.5 Expresiones

En muchas ocasiones, la ejecución de una tarea del algoritmo implica la realización de un cálculo matemático (operaciones aritméticas y/o lógicas). Una **expresión** es una serie de términos (constantes, literales, variables y funciones) posiblemente agrupados mediante paréntesis y conectados mediante operadores (aritméticos como +, - y lógicos como **&**, **|**), que representan un cálculo matemático.

El proceso que permite determinar el valor de la expresión, es decir el resultado del cálculo, es conocido como **evaluación de expresión**. Según el tipo del resultado de la expresión, el cual es conocido como **tipo de la expresión**, las expresiones se clasifican en:

- **Expresiones numéricas:** Si el resultado de la expresión es un entero o un real.
- **Expresiones lógicas:** Si el resultado de la expresión es un valor de verdad.

4.1.5.1 Expresiones numéricas

Son expresiones en las que solamente aparecen operadores aritméticos (+, -, *, /) , conectando términos de tipo numérico exclusivamente. Por ejemplo, una expresión como:

(A + 5) * (Y + piso(X+2.5))

Es una expresión numérica si **A** es una variable de tipo entero (pues **A** esta siendo sumada con un literal de tipo entero **5**), **Y** es una variable de tipo entero y **X** es una variable de tipo real (la función **piso(X)** calcula el entero más cercano al real **X** por debajo, por ejemplo, piso(3.45) = 3 y piso(-4.678) = -5).

Entre las funciones numéricas y de manejo de cadenas más usadas en programación y definidas en el pseudocódigo se cuentan:

Funciones nombre_funcion(<parametros>): <tipo>	**Descripción de la Función**
raiz2(x:real):real	Calcula raiz cuadrada
ln(x:real):real	Calcula logaritmo natural
exp(x:real):real	Calcula e^x
sen(x:real):real	Calcula seno (x)
cos(x:real):real	Calcula coseno(x)
tan(x:real):real	Calcula tangente(x)
asen(x:real) :real	Calcula arc seno(x)
acos(x:real) :real atan(x:real) :real abs(x:real) :real aleatorio(x:entero) :real elevar(x:real,exp:real) :real	Calcula arc coseno(x) Calcula arc tangente(x) Calcula valor absoluto de x Produce un numero aleatorio entero entre [0,x) Eleva x^{exp}
piso(x:real) :real	devuelve el entero menor más cercano a x
techo(x:real) :real	devuelve el entero mayor más cercano a x
enteroAreal(x:entero) :real	convierte un literal entero a real, o una variable de tipo entero a real
realAentero(x:real) :entero	convierte un literal real a entero, o una variable de tipo real a entero.
caracterAentero(c:caracter):entero	convierte un literal carácter a su representación en ASCII, o una variable de tipo carácter a su representación en ASCII
enteroAcaracter(x:entero):entero	convierte un literal entero a su carácter equivalente, o una variable de tipo entero a su carácter equivalente

longitudCadena(cad1: arreglo[] de caracter):entero	devuelve la longitud de la cadena.cad1
compararCadenas(cad1: arreglo[] de **caracter**, cad2: arreglo[] de caracter):**entero**	Compara las cadenas cad1 y cad2 en orden lexicográfico. devuelve un entero <0 si (cad1 < cad2) devuelve un entero >0 si (cad1 > cad2) devuelve 0 si (cad1 = cad2)
compararCadenasM(cad1: arreglo[] de **caracter**, cad2: arreglo[] de caracter):**entero**	Compara las cadenas cad1 y cad2 en orden lexicográfico, sin importar las letras en mayúscula. devuelve un entero <0 si (cad1 < cad2) devuelve un entero >0 si (cad1 > cad2) devuelve 0 si (cad1 = cad2)

4.1.5.2 Expresiones lógicas

Una expresión lógica es aquella en la cual el resultado se da en términos de **verdadero** o **falso**. Generalmente una expresión lógica se construye a partir de expresiones comparativas (dos expresiones numéricas relacionadas mediante algún operador relacional[3]), de variables y/o literales booleanos, los cuales se conectan mediante operadores lógicos.

Los operadores de comparación usados en programación son los siguientes: mayor que (>), menor que (<), igual (=), mayor o igual que (>=), menor o igual que (<=) y diferente (<>); mientras que los operadores lógicos usados son: o (•), y (**&**) y la negación (~). En el siguiente cuadro se resumen las tablas de verdad de estos operadores lógicos.

TABLAS DE VERDAD

Negación : (~)

p	~(p)
V	F
F	V

Conjunción lógica (&) y disyunción lógica (•)

p	q	p | q	p&q
V	V	V	V
V	F	V	F
F	V	V	F
F	F	F	F

Por ejemplo, una expresión como:

(A + 5 >=3) & (Y + piso(X + 2.5) <=3 * X) • ~bandera

Es una expresión lógica si **A** y **Y** son variables de tipo entero, **X** es una variable de tipo real y **bandera** es una variable de tipo booleano.

4.1.5.3 Evaluacion de expresiones

Una expresión sea del tipo que sea se evalúa mediante el siguiente algoritmo:

Inicio
PASO 1. Reemplazar todas las variables de la expresión por su valor.
PASO 2. Desde los paréntesis más internos hacia los más externos mientras existan paréntesis y/o operadores hacer:
2.1. Si una función no tiene todos sus argumentos evaluados, evaluar cada uno de los mismos.
2.2. Evaluar toda función que tenga sus argumentos evaluados y reemplazarla por su valor resultado.
2.3. Realizar las operaciones indicadas según la precedencia de los operadores que actúan sobre números y/o valores de verdad, es decir, términos ya evaluados.
2.4. Si sólo un número y/o valor de verdad se encuentra entre paréntesis eliminar los paréntesis.
PASO 3. El número o valor de verdad que resulta es el valor de la expresión.
Fin

Para que una expresión sea correcta, los términos que se conectan mediante un operador deben ser del mismo tipo. Por ejemplo, si en una expresión se suma una variable A con un literal entero, la variable debe ser tipo entero.

- ***Precedencia de operadores***

La precedencia de operadores es un orden de evaluación estándar, que se le ha asignado a los operadores para evitar excesivo uso de paréntesis. Evitan las posibles ambigüedades en el proceso de evaluación, es decir, evitan que una expresión pueda tomar más de un valor.

Por ejemplo la expresión 3 + 4 * 5, es ambigua pues puede ser interpretada como: (3 + 4) * 5 = 35 o como 3 + (4 * 5)=23, las cuales al ser evaluadas producen resultados diferentes.

La precedencia de operadores está definida en la siguiente tabla:

Precedencia de operadores	
Precedencia	**Operadores**
8	**( ) Paréntesis**
7	**- (signo menos)**
6	***, /, mod**
5	**+, - (substracción)**
4	**>, <, = ,> = , < =, < >**
3	**~**
2	**&**
1	**|**

Entre más alta es la precedencia más rápido se debe evaluar el operador.

Nótese que los paréntesis tienen la máxima precedencia, es decir, lo que está entre los paréntesis es lo primero que se evalúa; mientras que la disyunción lógica (|) tiene la más baja precedencia, lo que indica que será el último en ser evaluado. Cuando una expresión esta formada por operadores de la misma precedencia, dichos operadores son evaluados de izquierda a derecha, salvo unas pocas excepciones.

El operador **mod**, retorna el módulo (o residuo) de la división entera de dos números, por ejemplo, **11 mod 3 = 2** y **11 mod 4 = 3**.

- ***Ejemplos de evaluación de expresiones***

Si los valores de las variables enteras **A**, **B** y **C** son 5, 3 y 9 respectivamente, la siguiente expresión **-(3 + 4) * A + B * C** se evalúa de la siguiente forma:

-(3 + 4) * 5 + 3 * 9	Reemplazando las variables por sus valores
-(7)*5+3*9	Paréntesis prioridad más alta (8)
-7*5+3*9	Eliminación paréntesis
-35+27	Multiplicación, prioridad seis (6)
-8	Suma, prioridad cinco (5)

Si los valores de las variables enteras **A**, **B** y **C** son 5, 3 y 9 respectivamente, la siguiente expresión **-(3 + 4) * A / B * C + B * C** se evalúa de la siguiente forma:

-(3 + 4) * 5 / 3 * 9 + 3 * 9	Reemplazando las variables por sus valores
-(7) * 5 / 3 * 9 + 3 * 9	Paréntesis prioridad más alta (8)
-7 * 5 / 3 * 9 +3 * 9	Eliminación paréntesis
-35 / 3 * 9 +27	Multiplicación prioridad seis (6) de izquierda a derecha.
-11 * 9 +27	División prioridad seis (6) de izquierda a derecha.

-99 +27	Multiplicación prioridad seis (6).
-72	Suma prioridad cinco (5)

Si **A** y **Y** son variables de tipo **entero** y **X** de tipo **real**, con valores **6**, **8** y **–1.8** respectivamente, la expresión **(A + 5) * (Y+ piso(X+2.5))** se evalúa de la siguiente manera:

Reemplazar todas las variables por su valor	**(6+5)*(8+piso(-1.8+2.5))**
Si una función no tiene todos sus argumentos evaluados (los cuales pueden ser a su vez expresiones), evaluar cada uno de los mismos	**(6 + 5) * (8+ piso(0.7))**
Aplicar toda función que tenga todos sus argumentos evaluados y reemplazar la función por el valor resultado de la función.	**(6 + 5) * (8+ 0)**
Realizar las operaciones indicadas según la precedencia de los operadores que actúan sobre números y/o valores de verdad, es decir, términos ya evaluados.	**(11)*(8)**
Si sólo un número o un valor de verdad se encuentra entre paréntesis eliminar los paréntesis.	**11*8**
Realizar las operaciones indicadas según la precedencia de los operadores que actúan sobre números y/o valores de verdad, es decir, términos ya evaluados.	**88**
El número o valor de verdad que resulta es el valor de la expresión.	**88**

Sean **A** y **B** variables de tipo entero con valores **6** y **8** respectivamente, la expresión siguiente **(A > B+4) | ((A = 3) & ~(B < 4))** es evaluada como se muestra en la siguiente página.

Reemplazar todas las variables por su valor.	**(6>8+4) • ((6=3) & ~(8<4))**
Evaluar la suma	**(6>12) • ((6=3) & ~(8<4))**
Evaluar los operadores >, = y <	**(falso) • ((falso) & ~ (falso))**
Eliminar los paréntesis que encierran un solo valor.	**falso •(falso &~falso)**
Evaluar la negación (~).	**falso •(falso &verdadero)**
Evaluar la conjunción (&).	**falso •(falso)**
Eliminar los paréntesis que encierran un solo valor.	**falso • falso**
Evaluar la disyunción (•)	**falso**

Como se puede apreciar en el primer ejemplo, la división de enteros da como resultado un número entero y no un número real.

- ***Ejercicios de expresiones***

1. Si *tiempo*, *velocidad* y *peso* son variables de tipo real; *contador*, *itera* y *suma* son de tipo entero; *letra* y *primo* son de tipo carácter y *bandera* es de tipo booleano,

determinar cuáles de las siguientes expresiones son válidas y cuales no; se debe justificar la respuesta.

a. (tiempo + itera) / (velocidad + peso)
b. **piso(**tiempo**)** + contador < itera ***** suma & ~bandera
c. tiempo ***** velocidad < bandera & peso >=0.0
d. letra + contador ***** suma
e. **techo(**velocidad**) *** suma + **piso(**tiempo**) *** itera

2. Si *tiempo*, *velocidad* y *peso* son variables de tipo real; *contador*, *itera* y *suma* son de tipo entero; *letra* y *primo* son de tipo carácter y *bandera* es de tipo booleano, con valores 3.0, -4.5, 8.0, 5, -2, 30 y falso respectivamente, evaluar las siguientes expresiones

a. itera + **piso(**tiempo + velocidad**)** / (suma + itera)
b. letra <>primo **& techo(**tiempo**)** / **piso(**velocidad**)** + cuenta < 5
c. peso ***** velocidad **/ (** tiempo + 5.6 **)**
d. contador + (**(**itera**)*(itera)*** suma**)** / 4 – itera
e. bandera **&**(**raiz2(**tiempo**)** <=peso ***** velocidad)
f. contador < suma • tiempo > 1.0 **&** ~(suma = itera)

4.1.5.4 Instrucciones sobre variables, literales y constantes

Las dos operaciones que se pueden realizar sobre una variable son modificar su valor y mostrar su valor. Existen dos instrucciones diferentes que permiten modificar el valor de una variable: asignación y lectura, mientras que existe sólo una forma de mostrar el valor: escritura.

La única operación que se puede realizar sobre constantes y/o literales es mostrar su valor y se usa de la misma manera como se muestra el valor de una variable.

- ***Asignación***

La **asignación** es la instrucción por medio de la cual se puede modificar el valor de una variable utilizando el resultado de una expresión. Para que la asignación sea válida, es decir, se pueda realizar, la variable y la expresión deben ser del mismo tipo.

La asignación tiene la siguiente forma:

<**nombre**>:=<**expresión**>

Donde, **<nombre>** es el nombre de la variable, que debe ser previamente declarada, y <**expresión**>es una expresión del mismo tipo de la variable **<nombre>**.

El símbolo := se usará para indicar que el valor de la expresión <**expresión**> se almacena en la variable <**nombre**>.

Para realizar una asignación (ejecutar), se sigue el siguiente algoritmo:

1. Evaluar la expresión.
2. Almacenar el resultado de la expresión en la dirección de memoria referenciada por la variable.

Ejemplos de asignación

Si **n** es una variable de tipo entero que tiene el valor **5**, la asignación **n :=n + 1,** se ejecuta de la siguiente manera:

1.	Evaluar la expresión (1. reemplazar todas la variables por su valor)	**n :=5 + 1**
1.	Evaluar la expresión (2c. realizar las operaciones indicadas según su precedencia)	**n :=6**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**n = 6**

La variable **n** queda finalmente con el valor de 6.

Si **n**, **m** y **k** son variables de tipo entero y tienen los valores **5**, **4** y **–3** respectivamente, la asignación **m :=k + n * 5**, se ejecuta de la siguiente manera:

1.	Evaluar la expresión (1. Reemplazar todas la variables por su valor)	**m := -3 + 5 * 5**
1.	Evaluar la expresión (2c. realizar las operaciones indicadas según su precedencia)	**m :=22**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**m = 22**

La variable **m** queda finalmente con el valor de **22**.

Si **A** y **Y** son variables de tipo **entero** y **X** de tipo **real**, con valores **6**, **8** y **–1.8** respectivamente, la asignación **A :=(A + 5) * (Y+ piso(X+2.5))**, es ejecutada de la siguiente manera:

1.	Evaluar la expresión (La evaluación fue realizada en los ejemplos de la sección anterior)	**A :=88**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**A = 22**

La variable **A** queda con el valor **22**.

Sean **A** y **B** variables de tipo entero con valores **6** y **8** respectivamente, **bandera** de tipo booleano con valor **verdadero**, la asignación: **bandera :=(A > B+4) • ((A = 3) &~(B < 4))** es ejecutada como se muestra en la página siguiente.

1.	Evaluar la expresión (La evaluación fue realizada en los ejemplos de la sección anterior)	**bandera :=falso**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**bandera = falso**

La variable **bandera** queda con el valor de verdad **falso**.

Suponga que **y** es una variable de tipo real que tiene el valor –3.0, la asignación **y :=3 + y * y** es ejecutada como se muestra en la siguiente página.

1.	Evaluar la expresión	**y :=12.0**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**y = 12.0**

La variable **y** queda finalmente con el valor de **12.0**.

Como se puede apreciar en los ejemplos anteriores, la variable a la que se le está asignando el resultado de la expresión puede aparecer en la expresión, sin que ello sea un error. Se debe recordar que primero se evalúa la expresión (tomando el valor que tengan las variables en ese momento), y posteriormente se almacena el resultado en la variable.

- ***Lectura y escritura***

La instrucción **lectura** es la que le permite al usuario del algoritmo modificar el valor de una variable, durante la ejecución del mismo. Esta instrucción espera que el usuario ingrese un valor (mediante el teclado el usuario escribe el valor a almacenar en la variable) y después, garantizando que sea del tipo apropiado, lo asigna a la variable. La lectura tiene la siguiente forma:

En este curso una variable es leída de la siguiente manera:

leer (<nombre>)

donde, **<nombre>** es el nombre de la variable que se está leyendo.

Ejemplos de lectura

- leer (n)
- leer (m)
- leer (x)
- leer (y)

Si **x, y** son variables de tipo real y **n**, **m** son variables de tipo entero y si el usuario en el primer ejemplo ingresa el valor de **30**, después de ejecutar la instrucción de lectura la variable **n** tendrá el valor de **30**.

La **escritura** es la instrucción que permite mostrar, normalmente en la pantalla, el

valor de una variable, una constante o un literal durante la ejecución . La escritura tiene la siguiente forma:

escribir(<nombre>)

donde, <nombre> es el nombre de la variable, constante o literal que se está mostrando.

Ejemplos de escritura

- escribir (n)
- escribir (m)
- escribir (x)
- escribir (y)
- escribir (cambioLinea)
- escribir ("Hola, buenos días")

Si **x, y** son variables de tipo real y **n, m** son variables de tipo entero, en el primer ejemplo si la variable **n** tiene el valor de **30**, en la pantalla se presentará dicho valor. En el último ejemplo en la pantalla aparecerá el texto **Hola, buenos días**. En el quinto ejemplo se está escribiendo un cambio de línea lo que tiene como efecto que lo siguiente que se escriba aparezca en la siguiente línea de la pantalla del computador.

Codigo

```
procedimiento principal()
variables
   // declaración de una variable de tipo booleano
   miBooleano: booleano
   // declaración de una variable de tipo entero
   miEntero: entero
   // declaración una variable de tipo real
   miReal: real
   // declaración de una variable de tipo caracter
   miCaracter: caracter
   // declaración de una variable de tipo cadena
   miCadena: cadena

 inicio
   // inicializar miBooleano
   miBooleano := verdadero
   // inicializar miEntero
   miEntero := 2
   // inicializar miReal
   miReal := 0.0
   // inicializar miCaracter
   miCaracter := 'c'
   // inicializar miCadena
```

```
   miCadena := "Respuesta: "
   escribir("Valores de mis variables: ")
   cambioLinea()
   escribir("variable miBooleano:  ")
   escribir(miBooleano)
   cambioLinea()
   escribir("variable miEntero:  ")
   escribir(miEntero)
   cambioLinea()
   escribir("variable miReal:  ")
   escribir(miReal)
   cambioLinea()
   escribir("variable miCaracter:  ")
   escribir(miCaracter)
   cambioLinea()
   escribir("variable miCadena:  ")
   escribir(miCadena)
   cambioLinea()
fin-procedimiento
```

- ***Ejercicios de instrucciones sobre variables***

1. Si *x*,*v* y *p* son variables de tipo real; *cont*, *i* y *k* son variables de tipo entero; *letra* y *c* son variables de tipo carácter, *band* y *terminar* son variables de tipo booleano; *MAX* es una constante de tipo entero y *PI* es una constante de tipo real, determinar cuáles de las siguientes instrucciones son válidas. Se debe justificar la respuesta.

- leer (letra)
- escribir (MAX)
- leer (cont)
- escribir (x)
- leer (MAX)
- leer (v)
- x := enteroAreal(k) + 2.0 * PI
- letra :='p'
- letra :="p"
- letra :=p
- letra :=c
- p :=v / x * raiz2(p) – band
- x :="3.2444" + "1.4e-4"
- x :=3.2444 + 1.4e-4
- x :='v' + 'p'
- terminar :=p > v >=x
- terminar :=verdadero
- terminar :="falso"
- terminar :='f' <=letra
- x + 3.0 :=v
- v :=x * p + enteroAreal(caracterAentero(letra))
- band :=(x+v)*p | terminar | cont <= k + i
- k := caracterAentero (letra + caracterAentero (c)) + k

2. Si *x*,*v* y *p* son variables de tipo real, con valores 3.5, 1.4 y 6.0 respectivamente, determinar el valor de la variable *z* de tipo real al realizar cada una de las asignaciones siguientes:

- z :=v - p / (x + p / x – v)
- z :=p / x / v + p / x * v
- z:= (v – 3.0 * x / p) / (4.0 – v / (5.0 + p / x))
- z :=2.0*p – 4.0/v + 5.0*x/(3.0+v*p – x)

3. Suponga que *x, v* y *p* son variables de tipo real, con valores 3.5, 1.4 y 6.0 respectivamente; *i, k* y *cont* son variables enteras con valores 5, 2 y –4, respectivamente; *letra* y *c* son variables de tipo carácter con valores 'p' y 't'; y *bandera* y *terminar* de tipo booleano con valores falso y verdadero respectivamente. Determine el valor de cada una de las variables, después de ejecutar las siguientes instrucciones, en el orden en que aparecen.

- x :=v + enteroAreal(caracterAentero(letra))
- k :=(k–2*cont)* (k–2*cont) + i*(cont – k / 2.0)/cont
- v :=x + (p – raiz2(x) * (v – x) + 3.0) * p
- bandera :=(terminar •¬(i+2 <>cont •p>=v)) •(x+v) = p
- cont :=cont + 1
- i:= cont * realAentero(x * v – p/2.0) + k – i
- p :=p + x – enteroAreal(piso(p+x) + 5)
- letra :=enteroAcaracter(k)
- k :=(k + cont) mod (piso(p) + techo(v))

4. Suponga que *x, v* y *p* son variables de tipo real, con valores 3.5, 1.4 y 6.0 respectivamente; *i, k* y *cont* son variables enteras con valores 5, 2 y –4, respectivamente; *letra* y *c* son variables de tipo carácter con valores 'p' y 't'; y *bandera* y *terminar* de tipo booleano con valores falso y verdadero respectivamente. Determinar el valor de cada una de las variables después de ejecutar las siguientes instrucciones, en el orden en que aparecen.

- v :=x + (p – raiz2(x) * (v – x) + 3.0) * p
- bandera :=(terminar •~(i+2 <=cont •p>=v)) •(x+v = p)
- i :=cont * realAentero(x * v – p/2.0) + k – i
- k :=(k–2*cont) (k–2*cont) + i*(cont – k / 2.0)/cont
- x :=v + enteroAreal(caracterAentero(letra))
- cont :=cont + 1
- letra :=enteroAcaracter(k)
- k :=(k + cont) mod (piso(p) + techo(v))
- p :=p + x – enteroAreal(piso(p+x) + 5)

4.2 ESTRUCTURAS DE CONTROL

Las estructuras de control tienen como finalidad ir señalando el orden en que tienen que sucederse los pasos de un algoritmo.

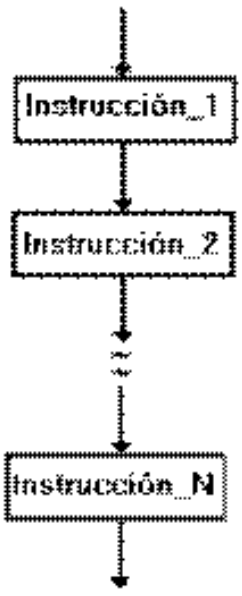
Ejemplo
Suponga que acaba de mostrar un mensaje en la pantalla que pregunte al usuario: ***"¿desea salir? - Si / No".***
Obviamente, de la respuesta del usuario va a depender la siguiente acción del programa. Por ejemplo, este mensaje se nos presenta tras haber seleccionado la opción de cerrar el programa, si nosotros digitamos "Sí", el programa finalizará, y si digitamos "No", el programa continuara con su ejecución. *El programador tiene que haber escrito código para las dos posibilidades, aunque cuando el programa esté funcionando, sólo se elegirá una.*

Las estructuras de control pueden clasificarse en tres tipos: Secuénciales, Selectivas y Repetitivas

4.2.1 Estructuras Secuenciales

Dada una lista de instrucciones **<instrucción 1> <instrucción 2> ... <instrucción n>** las estructuras **SECUENCIALES** permite la ejecución de dicha lista en el orden en que aparecen las instrucciones, es decir, se ejecuta primero la instrucción **<instrucción 1>** luego la instrucción **<instrucción 2>**, y por último se ejecuta la instrucción **<instrucción n>.** Una secuencia de instrucciones es llamada generalmente **BLOQUE DE INSTRUCCIONES**.

La forma general de la secuencia es:

DIAGRAMA DE FLUJO	PSEUDOCODIGO
	<instrucción1> <instrucción2> . . . <instrucción n>

aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las tres últimas líneas, teniendo en cuenta los valores para las constantes y las variables.

LINEA	r	a	ENTRADA	SALIDA
4	5.0		5.0	
5		78.53981		
6				El área es:
7				78.53981

2. Un granjero tiene una cantidad **X** de animales de dos tipos: conejos y gansos. Si la cantidad de patas total de los animales **Y** es conocida, ¿cuántos conejos y cuantos gansos tiene el granjero?

ANALISIS DEL PROBLEMA

VARIABLES CONOCIDAS	La cantidad total de animales **X** y la cantidad de patas totales **Y**.
VARIABLES DESCONOCIDAS	La cantidad de conejos y la cantidad de gansos.
CONDICIONES	La suma de los conejos y los gansos es igual a **X**. La suma de los patas de los conejos (cuatro por cada uno) y de los gansos (dos por cada uno) es igual a **Y**.

ESPECIFICACIÓN

ENTRADAS	X, Y ∈ Naturales, (**X** es el total de animales, **Y** es el total de patas).
SALIDAS	C, G ∈ Naturales, (**C** es el número de conejos, **G** es el número de gansos).
CONDICIONES	X = G + C, Y = 2*G + 4*C

Aplicando los conocimientos de álgebra a las condiciones anteriores, para expresar las variables de salida **C** y **G** en términos de las variables de entrada **X** y **Y**, se tiene que:

$$G = (4*X - Y)/2,\ C = (Y - 2*X)/2$$

DISEÑO

Primera División:

Inicio
Paso 1. Leer las entradas
Paso 2. Realizar el cálculo
Paso 3. Imprimir los resultados
Fin

Segunda División:

```
Inicio
Paso 1.   Leer las entradas
Paso 1.1. Leer el número total de animales
Paso 1.2. Leer el número total de patas
Paso 2.   Realizar el cálculo
Paso 2.1. Calcular el número de gansos
Paso 2.2. Calcular el número de conejos
Paso 3.    Imprimir los resultados
Paso 3.1. Imprimir el número de gansos
Paso 3.2. Imprimir el número de conejos
Fin
```

División Final:

```
1  x: entero /* se define la variable para el número total de animales */
2  y: entero /* se define la variable para el número total de patas de los
3  animales */
4  g: entero /*se define la variable para el número de gansos*/
5  c: entero /*se define la variable para el número de conejos*/
6  leer (x) /* lee el número total de animales */
7  leer (y) /* lee el número total de patas de los animales */
8  g := (4*x - y) / 2 /* calcula el número de gansos */
9  c := (y - 2*x) / 2 /* calcula el número de conejos */
10 escribir ("Cantidad de gansos:" )
11 escribir (g)
12 escribir ("Cantidad de conejos:" )
   escribir (c)
```

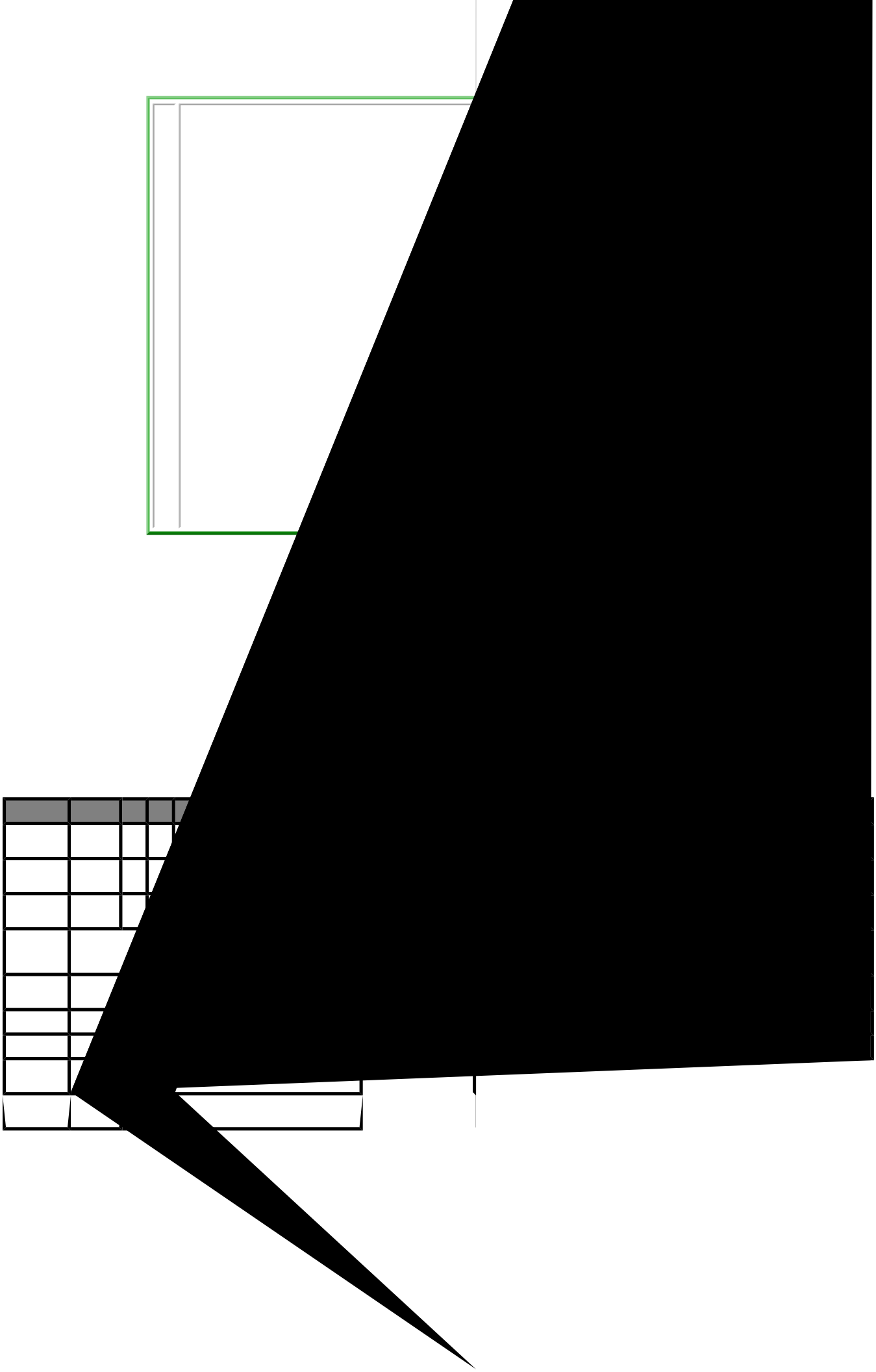
PRUEBA DE ESCRITORIO

Este algoritmo cuenta con doce (12) líneas, las cuatro primeras, son para definir las variables usadas y las últimas ocho, son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas (5-12), teniendo en cuenta los valores para las variables.

LINEA	X	y	g	c	ENTRADA	SALIDA
5	50				50	
6		140			140	
7			30			
8				20		
9						Cantidad de gansos:
10						30
11						Cantidad de conejos:

12					20

1



7			5			
8	La condición es evaluada a falso, por lo tanto se pasa a la línea 11, la siguiente línea al sino de la selección en ejecución.					
11				no es palíndrome		
12	Se salta a la siguiente línea.					
13						El número es:
14						no palíndrome

Como se puede apreciar en este ejemplo, una selección puede contener (en cualquiera de sus dos bloques de instrucciones), lecturas, escrituras, asignaciones u otras selecciones.

4.2.3 Estructuras de Selección Anidadas

Cuando una **<condición>** es muy compleja de plantear, se pueden realizar bloques de SI anidados, facilitando con esto el desempeño del compilador, evitando la evaluación de condiciones de dudosa eficiencia. Esta especificación se puede representar mediante el siguiente algoritmo:

```
si<condición1> entonces
<variable> := <expresión 1>
sino
  si <condición 2>entonces

      <variable> := <expresión 2>

  sino
 ..............
  sino
    si <condición n>entonces

        <variable>:= <expresión n>

   sino

        <variable> := <expresión
final>

   fin_si
.............
  fin_si

fin_si
```

Ejemplo de selección anidada.

Dados tres números enteros calcular el máximo.

```
1
2
3
4  a : entero /* se define la variable para el primer número */
   b : entero /* se define la variable para el segundo número */
   c : entero /*se define la variable para el tercer número */
5
6  mayor : entero /*se define la variable para el máximo */
7  leer (a)  /* lee el primer número */
   leer (b)  /* lee el segundo número */
   leer (c)  /* lee el tercer número */
8  si a >= b & a >= c entonces
9   mayor :=a  /* el máximo es a pues es mayor que b y c */
10 sino
11   si b >= a & b >= c entonces
12      mayor :=  b  /* el máximo es b pues es mayor que a y c */
13   sino
14      mayor :=  c /* sino es ni a ni b entonces es c */
15   fin_si
   fin_si
16 escribir  ("El máximo de los tres es:  ")
   escribir (mayor)
17
18
```

PRUEBA DE ESCRITORIO

Este algoritmo cuenta con dieciocho (18) líneas, las cuatro primeras, son para definir las variables usadas y las últimas catorce son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas 5-18, teniendo en cuenta los valores para las variables.

Primer prueba de escritorio

LINEA	a	b	C	mayor	ENTRADA	SALIDA
5	50				50	
6		140			140	
7			30		30	
8	La condición es evaluada a falso, por lo tanto se pasa a la línea 11, la siguiente al si_no de la selección en ejecución.					
11	La condición es evaluada a verdadero, por lo tanto se pasa a la línea 12, la siguiente línea al entonces de la selección en ejecución.					
12				140		
13	Se salta hasta el fin_si de la selección en ejecución, es decir, hasta la línea 15					
15	Se salta a la siguiente línea.					
16	Se salta a la siguiente línea.					
17						0 rg /TT0 1 Tf (mayor)Tj
18						

```
seleccionar  < opción> hacer
caso <constante_1>:
                    <bloque instrucciones 1>
caso <constante_2>:
                    <bloque instrucciones 2>
.............
caso <constante_n>:
                    <bloque instrucciones n>
otrocaso :
                    <bloque instrucciones>

fin_seleccionar
```

Ejemplo de Selección múltiple.

Una persona selecciona una transacción en un cajero automático

ANALISIS DEL PROBLEMA

OBJETOS DESCONOCIDOS	Que transacción desea realizar el usuario
OBJETOS CONOCIDOS	Los números correspondientes a las transacciones disponibles en el cajero.
CONDICIONES	El cajero debe ejecutar la transacción seleccionada por el usuario

ESPECIFICACIÓN

ENTRADAS	n ∈ Enteros entre 1 y 5, (n es el número ingresado por el usuario para seleccionar la transacción).
SALIDAS	Ejecución de una Transacción
CONDICIONES	El numero n seleccionado por el usuario debe corresponder a la transacción seleccionada por el usuario

DISEÑO

Primera División:

```
Inicio
Paso 1. Leer el número.
Paso 2. Determinar la transacción y
Ejecutarla.
Fin
```

Segunda División:

```
Inicio
Paso 1.  Leer el número.
Paso 2.  Determinar la transacción
Paso 2.1. si es 1
Paso 2.1.1. Ejecutar la transacción 1
Paso 2.1.2. Salir
Paso 2.2. si es 2
Paso 2.2.1. Ejecutar la transacción 2
Paso 2.2.2. Salir
Paso 2.3. si es 3
Paso 2.3.1. Ejecutar la transacción 3
Paso 2.3.2. Salir
Paso 2.4. si es 4
Paso 2.4.1. Ejecutar la transacción 4
Paso 2.4.2. Salir
Paso 2.5. si es 5
Paso 2.5.1. Ejecutar la transacción 5
Paso 2.5.2. Salir
Paso 2.5. si no esta en el rango
Paso 2.5.1. Salir
Fin
```

División Final:

```
n: entero /* se define la variable para la selección del usuario*/
leer (n)  /* lee el número de la transacción seleccionada*/
seleccionar  n hacer
caso 1:
        <consignación>

caso 2:
        <retiro>

caso 3:
        <pago de servicios>

caso 4:
        <cambio de clave>

caso 5:
        <consulta de saldo>

fin_seleccionar
```

---

[1] Declarar una variable es asignarle un nombre y definirle su tipo.

[2] En muchos casos se puede utilizar notación científica

[3] Formalmente se dice operador de comparación y no relacional. El nombre de relacional es dado por que estos operadores provienen de las **relaciones** de orden.