1. CONSTRUCTORES BÁSICOS

1.1. LOS DATOS EN PROGRAMACIÓN

1.1.1. *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 (").

1.1.2. *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.

1.1.2.1. 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

1.1.2.2. 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

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

1.1.2.3. *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.

1.1.3. *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).

1.1.3.1. *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

1.1.3.2. *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

1.1.3.3. *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.

[2] En muchos casos se puede utilizar notación científica

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.

1.1.3.4. 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 :***.

1.1.3.5. Literales booleanos

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

1.1.4. *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 = ' '

1.1.5. *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 **&**, **I**), 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.

1.1.5.1. 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	devuelve la longitud de la cadena.cad1

caracter)**:**entero	
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)

1.1.5.2. 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:

[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.

(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.

1.1.5.3. 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**