

2. ALGORITMOS

2.1. Introducción

La etapa vital de la solución de un problema con una computadora es el diseño del algoritmo y de la estructura fundamental de datos. Un algoritmo es un procedimiento expresado precisamente para obtener la solución del problema, la que se presenta de manera subsecuente a una computadora en el lenguaje de programación seleccionado. Los algoritmos se presentan de una manera conveniente para un lector humano, mientras que los programas sirven a las necesidades de las computadoras.

Es importante recordar mientras diseñamos un algoritmo que una computadora sólo sigue las instrucciones y no puede actuar si no se le ha ordenado de manera explícita. Por lo tanto, el solucionador de problemas debe prever cualquier aspecto del problema en el propio algoritmo.

La palabra **algoritmo** deriva del nombre de un matemático árabe del siglo IX, llamado Alkhuwarizmi, quien estaba interesado en resolver ciertos problemas de aritmética y describió varios métodos para resolverlos. Estos métodos fueron presentados como una lista de instrucciones específicas (como una receta de cocina) y su nombre se utiliza para referirse a dichos métodos.

2.2. Definición

Un algoritmo es, en forma intuitiva, una receta, un conjunto de instrucciones o de especificaciones sobre un proceso para hacer algo. Ese “algo” generalmente es la solución de un problema de algún tipo. Formalmente un algoritmo se puede definir de la siguiente forma:

Un algoritmo puede definirse como una secuencia ordenada de pasos elementales, exenta de ambigüedades, que lleva a la solución de un problema dado en un tiempo finito.

Para comprender la definición anterior se clarifica algunos conceptos.

Ejemplo: escriba un algoritmo que permita preparar una tortilla de papas de tres huevos.

Si la persona que resuelva el problema es un cocinero lo resuelve sin mayor nivel de detalle, pero si no lo es, se deben describir los pasos necesarios para realizarlo:

Paso 1: Mezclar papas fritas, huevos y una pizca de sal en un recipiente.

Paso 2: Freir.

Esto podría resolver el problema, pero si la persona que lo ejecute no sabe cocinar, se debe detallar, cada uno de los pasos mencionados, pues estos no son lo bastante simples para un principiante. De esta manera el primer paso se puede descomponer en:

Paso 1: Pelar las papas

Paso 2: Cortar las papas en cuadraditos

Paso 3: Freír las papas

Paso 4: Batir los huevos en un recipiente

Paso 5: Verter las papas en un recipiente y echar una pizca de sal

El tercer paso puede descomponerse en:

- Calentar el aceite en la sartén
- Verter las papas en la sartén
- Sacar las papas ya doradas en un recipiente

Nótese que si la tortilla la va a ejecutar un niño, alguna tareas, por ejemplo batir huevos, pueden necesitar una mejor especificación.

Con este ejemplo se pretende mostrar que la lista de **pasos elementales** que compongan nuestro algoritmo depende de quien sea el encargado de ejecutarlo. Si en particular, el problema va a ser resuelto utilizando una computadora, el conjunto de pasos elementales conocidos es muy reducido, lo que implica un alto grado de detalle para los algoritmos.

Se considera entonces como un paso elemental aquel que no puede volver a ser dividido en otros más simples.

Otro aspecto importante de un algoritmo es su nivel de detalle, que no debe confundirse con el concepto de paso elemental. En ocasiones, no se trata de descomponer una orden en acciones más simples sino que se busca analizar cuáles son las órdenes relevantes para el problema. Para comprender lo expuesto se lo analiza con un ejemplo:

Ejemplo: escriba un algoritmo que describa la manera en que Ud. se levanta todas las mañanas para ir al trabajo.

Paso 1: Salir de la cama

Paso 2: Quitarse el pijama

Paso 3: Ducharse

Paso 4: Vestirse

Paso 5: Desayunar

Paso 6: Arrancar el auto para ir al trabajo

La solución del problema se expresó en seis pasos, descartando aspectos que se suponen o sobreentienden, como “colocarse los zapatos al vestirse” o “abrir la puerta del auto” pues a nadie se le ocurriría ir a trabajar descalzo. En cambio existen aspectos que no pueden obviarse o suponerse porque el algoritmo perdería lógica, por ejemplo el paso “vestirse”, no puede ser omitido. Puede discutirse si se requiere un mayor nivel de detalle o no, pero no puede ser eliminado del algoritmo.

Un buen desarrollador de algoritmos deberá reconocer esos aspectos importantes y tratar de simplificar al mínimo su especificación de manera de seguir resolviendo el problema con la menor cantidad de órdenes posibles.

Además, en la definición de algoritmo se hace referencia a la ambigüedad y tiempo de respuesta, debido a que todo algoritmo debe cumplir con ciertas propiedades para que se lo considere como tal y proporcione el resultado deseado cuando un programa basado en él se presenta a una computadora.

Un algoritmo debe cumplir las siguientes propiedades:

Ausencia de Ambigüedad: si se trabaja dentro de cierto marco o contexto, la representación de cada paso de un algoritmo debe tener una única interpretación.

Ejemplo: indique la forma de condimentar una salsa.

Incorrecto: ponerle algunas especies a la salsa.

Correcto: ponerle sal, pimienta y orégano a la salsa.

Generalidad: un algoritmo se puede realizar para varios problemas que se relacionan entre si, es decir, se debe aplicar a un problema o clase de problemas específicos.

Ejemplo: indique la forma de marcar un número de teléfono.

Incorrecto: si la solución del algoritmo sirve para marcar solamente el número 4220234, solo tendrá valor para ese número.

Correcto: si la solución es un método para marcar cualquier número, entonces es útil. Por supuesto, debe haber alguna restricción a la generalidad de un algoritmo.

Tiempo de respuesta: la ejecución de un algoritmo debe finalizar después de que se haya llevado a cabo una cantidad finita de pasos. De otra manera, no se puede exigir que la ejecución produzca una solución.

Ejemplo: Llene la zanja con ese montón de arena

Algoritmo: tome una pala y empiece a echar arena en la zanja. Cuando se llene la zanja deténgase.

(se está seguro que en algún momento parará, aunque no se sabe cuanto tardará).

2.3. Dominio de un Algoritmo

La clase o el conjunto de datos y las condiciones para las cuales un algoritmo trabaja correctamente se llama dominio. Cuando se trata de resolver cualquier problema es necesario definir el dominio del algoritmo y después verificar que trabaja para todos los casos que se encuentran dentro de ese dominio.

Al decidir el dominio de un algoritmo es necesario incluir todas las situaciones similares, pero los casos remotos o poco probables se pueden omitir.

2.4. Errores en la Construcción de un Algoritmo

En la construcción de algoritmos se consideran dos tipos de errores:

- **Errores de Dominio:** se presentan cuando no se han especificado todas las situaciones que se pueden presentar en la práctica o se ha descuidado la apreciación de su importancia. Las pruebas más difíciles son aquellas que verifican que se ha seleccionado un dominio correcto para el algoritmo. Cuando una situación no prevista se presenta, hay tres opciones:
 - Ignorarla porque es improbable y quizás nunca ocurra.
 - Restringir el dominio del algoritmo para excluirla.
 - Corregir el algoritmo.
- **Errores de Lógica:** son aquellos errores que se detectan después de que se ha definido en forma adecuada el dominio de un algoritmo, en la etapa de prueba o verificación. se deben principalmente a las siguientes causas:
 - Etapas incorrectas
 - Secuencia incorrecta de etapas.

3. FORMAS DE EXPRESAR UN ALGORITMO

Un mismo algoritmo puede ser expresado de distintas formas.

- **Lenguaje común.** En el lenguaje normal que hablamos y escribimos; útil para comunicar un algoritmo a otra persona o en una fase de análisis previo de un sistema computacional.

- **Diagramas de flujo.** Es un lenguaje gráfico; útil para visualizar en forma rápida la secuencia lógica de pasos a seguir para un algoritmo y de gran ayuda para la traducción del mismo a un programa de computación
- **Pseudocódigo:** Es una técnica para expresar en lenguaje natural la lógica de un programa, es decir, su flujo de control. El pseudocódigo no es un lenguaje de programación sino un modo de plantear un proceso de forma que su traducción a un lenguaje de alto nivel sea sencillo para el programador.
- **Lenguajes de Programación:** Es la forma obligada de expresión de un algoritmo para que pueda ser leído, ejecutado y almacenado por el computador.



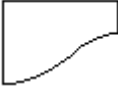

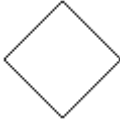



3.1. Diagramas de Flujo

Un diagrama de flujo es la representación gráfica o visual de un algoritmo. Se usan en el planeamiento, desarrollo y estructuración de un algoritmo. Mediante los diagramas de flujo el algoritmo se puede comunicar y documentar (porque enseña y describe el proceso).

Formalmente, un diagrama de flujo es un diagrama formado por símbolos (cajas, bloques, figuras) y flechas o líneas de flujo que conectan los símbolos entre si. Los símbolos denotan los pasos esenciales del algoritmo y las flechas indican la secuencia. Se dibujan de tal manera que la dirección del flujo sea hacia abajo o de izquierda a derecha.

3.2. Simbología Básica de los Diagramas de Flujo

Los símbolos que se describen a continuación, para la representación gráfica de los diagramas de flujo, son de uso universal.

SÍMBOLO	SIGNIFICADO
	Indica principio o fin de un algoritmo.
	Indica entrada de datos.
	Indica la salida de la información.
	Se usa generalmente para sentencias o enunciados de asignación (acción de asignar) y para la realización de un proceso matemático.
	Permite evaluar una expresión relacional ó lógica que puede tomar un valor verdadero o falso. En función de este resultado el flujo del algoritmo seguirá una determinada dirección.
	Se usa cuando el diagrama es largo y se requiere más de una hoja de papel o para evitar líneas que se crucen.
	Simbolizan módulos o segmentos lógicos
	Indica la dirección del algoritmo en cada momento mediante una flecha.

3.3. Utilidad de los Diagramas de Flujo

El diagrama de flujo refleja los pasos sucesivos que el computador debe dar para llegar a la solución de un problema. Las principales razones por las cuales es generalmente aconsejable el trazado de un diagrama de flujo son las siguientes:

- Oportunidad de verificar la lógica de la solución.
- Sirve de guía al programador para la codificación del programa.
- Permite fácilmente modificar un programa.
- Es útil para la discusión grupal.
- Sirve para documentar el programa.

ESTRUCTURAS DE CONTROL BÁSICAS

Al ser un algoritmo una secuencia de pasos ordenados, estos deben seguir una trayectoria para su ejecución desde el primer paso hasta el último. Esta trayectoria se denomina **flujo de control** que indica el orden en el cual deben ejecutarse los pasos elementales.

Para organizar el flujo de control de un algoritmo se utilizan **estructuras de control**, estas son construcciones algorítmicas lineales, de selección e iteración. Las dos últimas alteran el flujo de control lineal del algoritmo.

Las estructuras de control básicas para organizar el flujo de control en un algoritmo, son las siguientes:

- Estructura secuencial
- Estructura de selección
- Estructura de iteración

5.1. Estructura Secuencial

La estructura de control más simple está representada por una sucesión de acciones que se ejecutan de arriba hacia abajo sin bifurcaciones, es decir, una acción a continuación de otra.

Ejemplo: escriba un algoritmo que describa la forma en que una persona se levanta todas las mañanas para ir al trabajo.

Paso 1: Salir de la cama

Paso 2: Quitarse el pijama

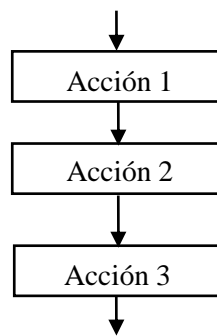
Paso 3: Ducharse

Paso 4: Vestirse

Paso 5: Desayunar

Paso 6: Arrancar el auto para ir al trabajo

A continuación se presenta gráficamente esta estructura.



5.2. Estructura de Selección

En un algoritmo representativo de un problema real es prácticamente imposible que las instrucciones sean secuenciales puras. Es necesario tomar decisiones en función de los datos del problema.

A través de la selección se incorpora, a la especificación del algoritmo, la capacidad de decisión. De esta forma será posible seleccionar una de **dos alternativas** de acción posibles durante la ejecución del algoritmo.

La selección se expresa con el siguiente pseudocódigo:

Si (condición)

entonces

acción o acciones a realizar si la condición es verdadera (1)

sino

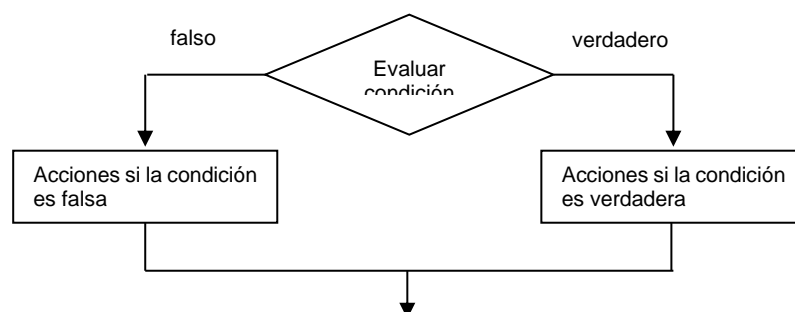
acción o acciones a realizar si la condición es verdadera (2)

FinSi

donde **condición** es una expresión que al ser evaluada puede tomar solamente uno de los dos valores posibles: verdadero o falso.

En el caso que la condición a evaluar resulte verdadera se ejecutarán las acciones (1) y **no** se ejecutarán las acciones (2). Si la condición a evaluar resulta falsa se ejecutarán las acciones (2) y **no** las acciones (1).

A continuación se presenta gráficamente esta estructura.



Puede ocurrir que no se tengan que representar acciones cuando la condición es falsa. En este caso se utilizará la siguiente notación:

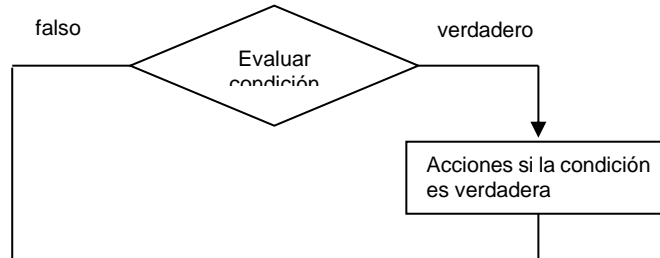
Si (condición)

entonces

acción o acciones a realizar si la condición es verdadera

FinSi

A continuación se presenta gráficamente esta estructura.



Si al evaluar un **decisión** toma más de dos valores, se utilizará la siguiente notación:

Si (*variable de decisión*)

Valor 1: Acción 1

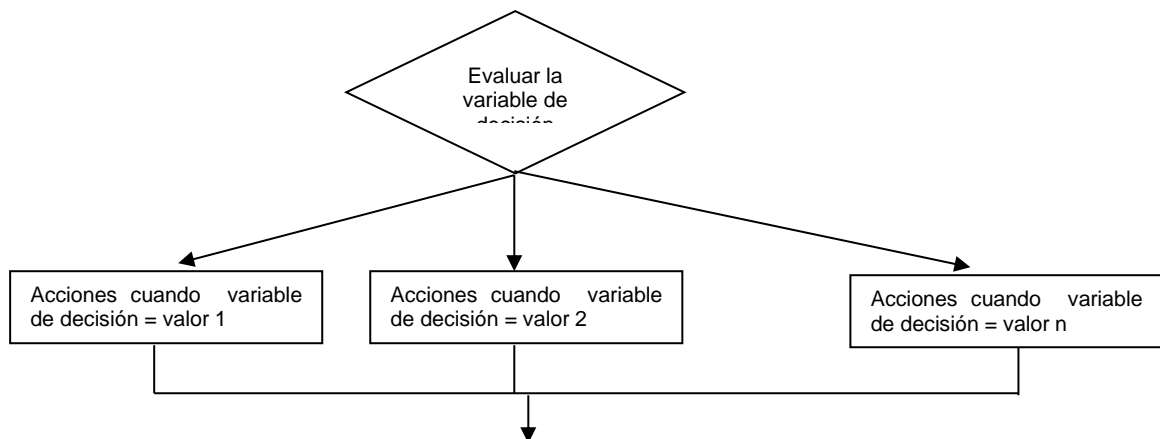
Valor 2: Acción 2

.....

Valor N: Acción N

[Otro: Acción N+1]

A continuación se presenta gráficamente esta estructura.



5.3. Estructura de Iteración

Una extensión natural de una estructura secuencial consiste en repetir N veces un bloque de acciones. El fin de la repetición dependerá de un valor predefinido o del cumplimiento de una determinada condición.

Existen dos formas de expresar esta estructura la *Repetición* y la *Iteración* en este curso solo utilizaremos la iteración.

Iteración

Puede ocurrir que se desee ejecutar un conjunto de acciones de un algoritmo desconociendo el número exacto de veces que se ejecutan. Para estos casos existen estructuras de control iterativas condicionales, es decir, las acciones se ejecutan dependiendo de la evaluación de una condición.

Por lo tanto, dentro de una estructura iterativa, además de una serie de pasos elementales que se repiten; es necesario contar con un mecanismo que lo detenga.

Podemos definir una estructura iterativa como la estructura de control que permite al algoritmo ejecutar en forma repetitiva un conjunto de acciones utilizando una condición para indicar su finalización.

Estas estructuras se clasifican en **pre-condicionales y pos-condicionales**.

Las estructuras pre-condicionales evalúan la condición y, si es verdadera, se ejecuta el conjunto de acciones; esto hace que dicho conjunto se puede ejecutar 0, 1 o más veces.

La notación para esta estructura es la siguiente:

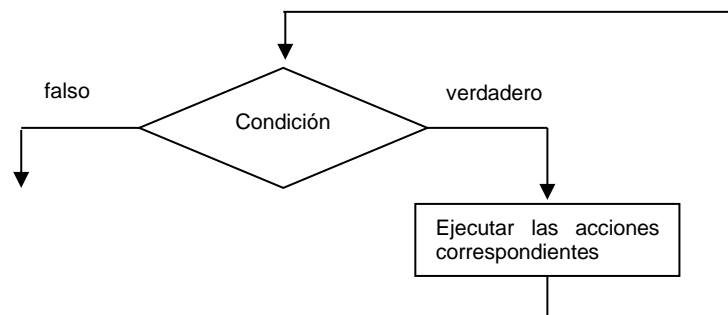
Mientras (condición)

Acción o acciones a realizar en caso de que la condición sea verdadera.

FinMientras

La condición es una expresión que sólo puede tener uno de dos valores posibles: verdadero o falso.

A continuación se presenta gráficamente esta estructura.



Las estructuras pos-condicionales, primero se ejecuta el conjunto de acciones, luego se evalúa la condición y, si es falsa, se ejecuta nuevamente el bloque de acciones. A diferencia de la estructura anterior iterativa anterior, el conjunto de acciones se debe ejecutar 1 o más veces. Nótese que, en este caso, el bloque de acción se ejecuta antes de evaluar la condición, por lo tanto se lleva a cabo al menos una vez.

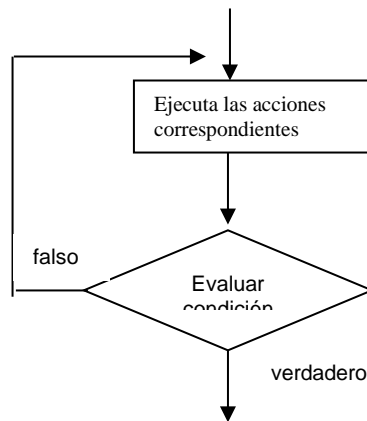
La notación a utilizar para esta estructura es la siguiente:

Repetir

Acción o acciones a realizar en caso de que la condición sea falsa

Hasta (condición)

A continuación se presenta gráficamente esta estructura.



6 TIPO DE DATOS

6.1. Datos Simples

Los programas que implementan los algoritmos necesitan alguna manera de representar los objetos del mundo real. Para ello los algoritmos operan sobre datos y estructuras de datos de distinta naturaleza, tales como números, letras, símbolos, etc.

Por lo tanto un **dato** es la expresión general que describe los objetos con los cuales opera una computadora. Los tipos de datos están ligados a un conjunto de operaciones que permiten su creación y manipulación.

Los tipos de datos se caracterizan por:

- Un rango de valores posibles
- Un conjunto de operaciones realizables sobre ese tipo
- Su representación

Los tipos de datos simples son:

- Numérico
- Lógico
- Carácter

6.1.1. Tipo de Dato Numérico

El **tipo de dato numérico** es el conjunto de los valores numéricos que pueden representarse de dos formas:

- Enteros
- Reales

Enteros

El tipo entero consiste de un subconjunto finito de los números enteros y su tamaño puede variar según el valor que tenga:

...-3, -2, -1, 0, 1, 2, 3...

Dado que una computadora tiene memoria finita, la cantidad de valores enteros que se pueden representar sobre ella son finitos, por esto se deduce que existe un número entero máximo y otro mínimo.

La cantidad de valores que se pueden representar depende de la cantidad de memoria (bits) que se utilicen para representar un entero.

Ejemplo:

Hay sistemas de representación numérica que utilizan 16 dígitos binarios (bits) para almacenar en memoria cada número entero, permitiendo un rango de valores enteros entre -2^{15} y $+2^{15}$. Otros sistemas utilizan 32 bits, por lo que el rango es entre -2^{31} y $+2^{31}$.

En general, para cada computadora existe el entero **maxint**, tal que todo número entero n puede ser representado si

$$-\text{maxint} \leq n \leq \text{maxint}$$

donde **maxint** identifica al número entero más grande que se puede representar con la cantidad de dígitos binarios disponibles.

Reales

El tipo de dato real es una clase de dato numérico que permite representar números decimales. Los valores fraccionarios forman una serie ordenada, desde un valor mínimo negativo, hasta un valor máximo determinado por la norma IEEE 754, de 1985, pero los valores no están distribuidos de manera uniforme en ese intervalo, como sucede con los enteros.

Se debe tener en cuenta que el tipo de dato real tiene una representación finita de los números reales; dicha representación tiene una precisión fija, es decir, un número fijo de dígitos significativos. Esta condición es la que establece una diferencia con la representación matemática de los números reales. En este caso se tienen infinitos números diferentes, en tanto que la cantidad de representaciones del tipo de dato real está limitada por el espacio en memoria disponible.

La representación para números reales se denomina **coma flotante**. Esta es una generalización de la conocida notación científica, que consiste en definir cada número como una

mantisa (o fracción decimal), que da los dígitos contenidos en el número; y un exponente (o característica), que determina el lugar del punto decimal con respecto a estos dígitos.

Ejemplo:

Sea el número 8941295000000000. Su representación científica en cualquier calculadora es $8,941295 \times 10^{16}$.

La mantisa del número real representado en la computadora en este caso es : 0,8941295 y el exponente 17. Nótese que, a diferencia de la representación anterior, la coma se ubica inmediatamente a la izquierda del primer dígito significativo y, por lo tanto, la magnitud del exponente se incrementa en 1.

Si el número es 0,0000000000356798, su representación utilizando notación científica es $3,56798 \times 10^{-11}$.

La mantisa es 0,356798 y el exponente es -10 .

Se observa que un exponente positivo lleva a desplazar la coma decimal tantos lugares hacia la derecha como lo indica su magnitud, mientras un exponente

6.1.2. Tipo de Dato “Carácter”

Un tipo de dato carácter proporciona objetos de la clase de datos que contiene solo un elemento como su valor. Este conjunto de elementos está establecido y normalizado por un estándar llamado ASCII (American Standard Code for Information Interchange), el cual establece cuales son los elementos y el orden de precedencia entre los mismos. Los elementos son las letras, número y símbolos especiales disponibles en el teclado de la computadora y algunos otros elementos gráficos. Cabe acotar que el código ASCII no fue único, pero es el más utilizado internacionalmente.

Son elementos carácter:

- Letras minúsculas: 'a', 'b', 'c', ..., 'y', 'z'
- Letras mayúsculas: 'A', 'B', 'C', ..., 'Y', 'Z'.
- Dígitos: '0', '1', '2', '3', ..., '8', '9'.
- Caracteres especiales tales como: '!', '@', '#', '\$', '%', ...

Se debe tener en cuenta que no es lo mismo el valor entero 1 que el símbolo carácter '1'. Un valor del tipo de dato carácter es **solo uno** de los símbolos mencionados.

Operaciones sobre Datos Carácter

Los operadores relacionales descriptos en el tipo de dato numérico, pueden utilizarse también sobre los valores del tipo de dato carácter. Esto es, dos valores de tipo carácter se pueden comparar por =, <>, >, <, >=, <=; el resultado de cualquiera de ellos es un valor de tipo de dato lógico.

La comparación de dos caracteres es posible dado que el código ASCII define para cada símbolo un valor en su escala. De esta manera, al comparar dos símbolos, para determinar el resultado se utiliza el valor dado por el código.

Dentro del código ASCII los valores de los dígitos son menores que los valores de las letras mayúsculas, y estos a su vez menores que los de las letras minúsculas. Los valores dentro del código para los símbolos especiales, o bien son menores que los dígitos, o bien mayores que las letras minúsculas.

Ejemplo:

(á = Á)

da como resultado *falso*

('c' < 'Z')

da como resultado *falso*

('c' < 'z')

da como resultado *verdadero*

('X' > '5')

da como resultado *verdadero*

(' ' < 'H')

da como resultado *verdadero*

observación: ' ' = espacio

representa un espacio en blanco

(' ' < 'H')

no puede evaluarse pues los operandos son de tipos

6.1.3. Tipo de Dato “Alfanumérico”, “String” o “Cadena de Caracteres”

En la mayoría de los lenguajes de programación existe un tipo estándar que es la cadena de caracteres(string).

De acuerdo a definiciones previas, un carácter es una representación determinada de una letra, número o símbolo que se guarda internamente de acuerdo a su representación determinada por el código ASCII. Cuando se trabaja con el tipo de dato string, se tienen n caracteres tratados como una única variable, donde n proviene de la definición del string.

Un tipo de dato String es una sucesión de caracteres que se almacenan en un área contigua de memoria y que puede ser leído o escrito.

El tipo de dato string debe indicar el tamaño máximo que se desea manejar, y se define como:

Type nombre-string = string [longitud]

Donde *nombre-string* es el identificador del tipo, y *longitud* es el número máximo de caracteres que puede contener.

Una vez definido el tipo se pueden declarar variables de ese tipo.

Ejemplos (en lenguaje Pascal)

```
Type cadena1 = string[10]
      cadena2 = string[25]
Var  c1, c2: cadena1
      denominación: cadena2
```

Ejemplos de asignación en variables string

```
c1 = 'pepe'
c2 = '678@#$abc'
denominación = 'remera deportiva'
```

6.1.4. Tipo de Dato “Lógico”

El tipo de dato lógico, también llamado booleano, en honor al matemático británico George Boole (quien desarrolló el Álgebra lógica o de Boole), es un dato que puede tomar un valor entre un conjunto formado por dos posibles:

- Verdadero (true)
- Falso (false)

Se utiliza en casos donde se representan dos alternativas a una condición. Por ejemplo, si se debe determinar si un valor es primo; la respuesta será verdadera (true) si

el número es divisible solamente por si mismo y la unidad; en caso contrario, si tiene algún otro divisor, la respuesta será falsa (false).

6.2. Constantes y Variables

Los algoritmos contienen ciertos valores que no deben cambiar durante la ejecución del mismo. Tales valores se llaman Constantes. De igual forma existen otros valores que cambian durante la ejecución del mismo, estas son las Variables.

Una **constante** es una partida de datos(objetos) que permanece sin cambios durante todo el desarrollo del algoritmo o durante la ejecución del programa.

Una **variable** es un objeto o partida de datos cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa.

6.3. Expresión: Definición

“Una expresión es un conjunto de variables y/o constantes unidas por operadores”.

Dependiendo del tipo de operadores con que se este trabajando y el resultado obtenido podremos hablar de *Expresiones Aritméticas* o *Expresiones Lógicas*.

Operadores Aritméticos

+	Suma
-	Resta
*	Multiplicación
/	División
^	Potencia

Una Expresión Aritmética es aquella que cuando se la evalúa siempre se obtiene un resultado numérico.

Ejemplos de expresiones aritméticas

$2 * 5 + 7 = 17$
 $3 * A / 4 + Cont \wedge 2$
 $Suma = Suma + N$

Operadores Relacionales

<	Menor
>	Mayor
<=	Menor igual
> =	Mayor igual
=	Igual a
<>	Distinto a

Operadores Lógicos

Los operadores lógicos o boléanos básicos son:

- Negación (Not)
- Conjunción (And)
- Disyunción (Or)

El resultado de estas operaciones es el correspondiente a las conocidas tablas de verdad.

Los operadores relacionales y lógicos mencionados anteriormente, que permiten comparar dos valores, dan como resultado un valor lógico.

Ejemplo:

(8 < 4)	da como resultado <i>falso</i>
(2.5 * 4 = 10)	da como resultado <i>verdadero</i>
(8 > 3)	da como resultado <i>verdadero</i>
not (8 > 3)	da como resultado <i>falso</i>
(7 > 2.4), (9 = 3)	son expresiones <i>verdaderas</i> y <i>falsa</i>
respectivamente	
(7 > 2.4) and (9 = 3)	una expresión que da un resultado <i>falso</i>

Para cada operador lógico se define un símbolo, los cuales se muestran en la tabla 2.1.

Operación	Operador	Simbolización
Conjunción	Y / AND	\wedge
Disyunción	O / OR	\vee
Negación	NO / NOT	\sim

Tabla 2.1. Conectores Lógicos

Tablas de verdad

Para poder analizar cualquier proposición compuesta y decir qué valor de verdad tiene, es usual hacerlo a través de lo que se conoce como tabla de verdad, la cual se define de la siguiente manera:

La tabla de verdad de una proposición es, como su nombre lo indica, una tabla donde se muestran todas las combinaciones posibles de los valores de verdad de dicha proposición.

Conjunción

Dadas dos proposiciones cualquiera p y q, la proposición molecular $p \wedge q$ representa la conjunción de p y q.

La conjunción de dos proposiciones es cierta únicamente en el caso en que ambas proposiciones lo sean.

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Disyunción

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Negación

Dada una proposición P su negación $\sim P$ permitirá obtener el valor de verdad opuesto.

El valor de verdad de la negación de una proposición verdadera es falso y el valor de verdad de la negación de una proposición falsa es verdadero.

p	$\sim p$
V	F
F	V

Ejemplos de Expresiones Lógicas

- $(8 < 4)$ da como resultado falso
- $(2.5 * 4 = 10)$ da como resultado verdadero
- $(8 > 3)$ da como resultado verdadero
- NOT $(8 > 3)$ da como resultado falso
- $(7 > 2.4)$ AND $(9 = 3)$ es una expresión que da un resultado falso
- $(7 > 2.4)$ OR $(9 = 3)$ es una expresión que da un resultado verdadero

Orden de Evaluación en la expresiones

Operador	Prioridad
NOT	Más alta (se evalúa primero)
*, /, AND	↓
+, -, OR	↓
<, >, <=, >=, =, <>	Más baja (se evalúa al último)
Si existen paréntesis, las expresiones de su interior se evalúan primero	