CONCEPTOS BÁSICOS SOBRE ALGORITMOS

PROPÓSITO DE APRENDIZAJE

En esta primera unidad se presenta una breve descripción del concepto y propiedades fundamentales de los algoritmos y de los programas, introduce al estudiante en los elementos básicos de un programa, tipo de datos, operaciones básicas, etc. Soportada por la mayoría de los lenguajes de programación, se muestran los métodos fundamentales para la resolución de problemas con computadoras y las herramientas de programación necesarias para ello. Se describen las etapas clásicas en la resolución de problemas, así como las herramientas clásicas tales como pseudocódigos, diagramas de flujo y diagramas N-S.

El flujo (orden en que se ejecutan las sentencias de un programa) es secuencial si no se específica otra cosa. Este tipo de flujo significa que las sentencias se ejecutan en secuencias, una después de otra, en el orden en que se sitúan dentro del programa. Para cambiar esta situación se utilizan las estructuras condicionales que permiten modificar el flujo secuencial del programa. Así, las estructuras condicionales se utilizan para seleccionar las sentencias que se han de ejecutar a continuación.

Un bucle o ciclo, en programación, es una sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada ha dicho bucle deje de cumplirse. Generalmente, un bucle es utilizado para hacer una acción repetida sin tener que repetir varias veces el mismo código, lo que ahorra tiempo, deja el código más claro y facilita su modificación en el futuro.

El propósito fundamental de esta unidad es el aprendizaje y diseño de algoritmos, introduciendo al estudiante en el concepto de algoritmo, programa, la resolución de problemas con computadora, las herramientas de programación, aplicación de los condicionales simples, compuestos y múltiples, ciclo para, ciclo repetir, ciclo mientras, contador, acumulador, centinelas.

CONCEPTOS CLAVES

Computadora, Hardware, UCP, Unidades de entrada y salida, memoria principal, bus del sistema, lenguajes máquinas, lenguaje ensamblador, lenguajes de alto nivel, Resolución de problemas, Análisis del problema, Diseño del algoritmo, Verificación de algoritmos, Fase de implementación, Dato, Entero, Real, Lógico, Carácter, Cadena, Constantes, Variables, Expresiones, Funciones, Diagramas de flujo, Diagrama Nassi-Schneiderman (N-S), Pseudocódigo, Comentarios, Palabras reservadas, Identificadores, Literales, Condicionales, Condicional simple, condicional compuesto, condicional anidado, condicional múltiple, Ciclo o bucle, ciclo para, ciclo repetir, ciclo mientras, contador, acumulador, centinela.

1. CONCEPTOS BÁSICOS SOBRE ALGORITMOS

La principal razón para que las personas aprendan lenguaje de programación es utilizar la computadora como una herramienta en la resolución de problemas. La resolución de un problema exige al menos los siguientes pasos:

1. Definición o análisis del problema
2. Diseño del algoritmo o método para resolverlo
3. Transformación del algoritmo en un programa
4. Ejecución y validación del programa

Uno de los objetivos fundamentales de este curso es el aprendizaje y diseño de algoritmos.

Este capítulo introduce al estudiante en el concepto de algoritmo y programa.

Escritura de algoritmos como secuencia de pasos.

Algoritmo.
La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

Tipos de Algoritmos

Cualitativos: Son aquellos en los que se describen los pasos utilizando palabras.

Cuantitativos: Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

Dos fases pueden ser identificadas en el proceso de creación de un programa:
- Fase de resolución del problema
- Fase de implementación (realización) en un lenguaje de programación.

La fase de resolución del problema implica la perfecta comprensión del problema, el diseño de una solución conceptual y la especificación del método de resolución del problema, el diseño de una solución conceptual y la especificación del método de resolución detallando las acciones a realizar mediante un algoritmo.

Fase de resolución del problema. Esta fase incluye, a su vez, el análisis del problema así como el diseño y posterior verificación del algoritmo.

Análisis del problema. El primer paso para encontrar la solución a un problema es el análisis del mismo. Se debe examinar cuidadosamente el problema a fin de obtener una idea clara sobre lo que se solicita y determinar los datos necesarios para conseguirlo.

Diseño del algoritmo. La palabra algoritmo deriva del nombre del famoso matemático y astrónomo árabe Al-Khôwarizmi (siglo IX) que escribió un conocido tratado sobre la manipulación de números y ecuaciones titulado Kitab al-jabr w'almugabala.

Un algoritmo puede ser definido como la secuencia ordenada de pasos, sin ambigüedades, que conducen a la solución de un problema dado y puede ser expresado en lenguaje natural, por ejemplo el castellano.

Todo algoritmo debe ser:

- *Preciso:* Indicando el orden de realización de cada uno de los pasos.
- *Definido:* Si se sigue el algoritmo varias veces proporcionándoles de los mismos datos, se debe obtener siempre los mismos resultados.
- *Finito:* Al seguir el algoritmo, éste debe determinar en algún momento, es decir tener un número finito de pasos.

Para diseñar un algoritmo se debe comenzar por identificar las tareas más importantes para resolver el problema y disponerlas en el orden en el que han de ser ejecutadas. Los pasos en esta primera descripción de actividades deberán ser refinados, añadiendo más detalles a los mismos e incluso, algunos de ellos, pueden requerir un refinamiento adicional antes que podamos obtener un algoritmo claro, preciso y completo. Este método de diseño de los algoritmos en etapas, yendo de los conceptos generales a los detalle a través de refinamientos sucesivos, se conoce como método descendente (top-down).

En un algoritmo se debe de considerar tres partes:

1. Entrada: Información dada al algoritmo
2. Proceso: Operaciones o cálculos necesarios para encontrar la solución del problema.
3. Salida: Respuestas dadas por el algoritmo o resultados finales de los cálculos.

Como ejemplo imagine que desea desarrollar un algoritmo que calcule la superficie de un rectángulo proporcionándole su base y altura. Lo primero que deberá hacer es plantearse y contestar a las siguientes preguntas:

Especificaciones de entrada

¿Qué datos son de entrada?
¿Cuántos datos se introducirán?

¿Cuántos son datos de entrada válidos?

Especificaciones de salida

¿Cuáles son los datos de salida?
¿Cuántos datos de salida se producirán?
¿Qué precisión tendrán los resultados?
¿Se debe imprimir una cabecera?

El algoritmo en el primer diseño se podrá representar con los siguientes pasos:

Paso 1. Entrada desde periférico de entrada, por ejemplo teclado, de base y altura.
Paso 2. Cálculo de la superficie, multiplicando la base por la altura.
Paso 3. Salida por pantalla de base, altura y superficie.

El lenguaje algoritmo debe ser independiente de cualquier lenguaje de programación particular, pero fácilmente traducible a cada uno de ellos.

Verificación de algoritmo. Una vez que se ha terminado de escribir un algoritmo es necesario comprobar que realiza las tareas para las que se ha diseñado y produce el resultado correcto y esperado.

El modo más normal de comprobar un algoritmo es mediante su ejecución manual, usando datos significativos que abarquen todo el posible rango de valores y anotando en una hoja de papel las modificaciones que se producen en las diferentes fases hasta la obtención de los resultados. Este proceso se conoce como prueba del algoritmo o prueba de escritorio.

Fase de implementación. Una vez que el algoritmo está diseñado, representado mediante un método normalizado (diagrama de flujo, diagrama N-S o pseudocódigo), y verificado se debe pasar a la fase de codificación, traducción del algoritmo a un determinado lenguajes de programación, que deberá ser completada con la ejecución y comprobación del programa en la computadora.

EJERCICIOS RESUELTOS

Desarrolle los algoritmos que resuelvan los siguientes problemas:

a. Hacer una taza de té

Análisis del problema.

DATOS DE SALIDA: Taza de té
DATOS DE ENTRADA: Bolsa de té, agua
DATOS AUXILIARES: Pitido de la tetera, aspecto de la infusión

Después de echar agua en la tetera, se pone al fuego y espera a que el agua hierva (hasta que suena el pitido de la tetera). Introducimos el té y se deja un tiempo hasta que está hecho.

Diseño del Algoritmo

Inicio
Tomar la tetera
Llenarla de agua
Encender el fuego
Poner la tetera en el fuego
Mientras no hierva el agua
Esperar
Tomar la bolsa de té
Introducirla en la tetera
Mientras no esté hecho el té
Esperar
Echar el té en la taza
Fin

b. Reparar un pinchazo de una bicicleta

Análisis del problema

DATOS DE SALIDA: La rueda reparada
DATOS DE ENTRADA: La rueda pinchada, los parches, el pegamento
DATOS AUXILIARES: Las burbujas que salen donde está el pinchazo

Después de desmontar la rueda y la cubierta e inflar la cámara, se introduce la cámara por secciones en un cubo de agua. Las burbujas de aire indicarán donde está el pinchazo. Una vez descubierto el pinchazo se aplica el pegamento y ponemos el parche. Finalmente se monta la cámara, la cubierta y la rueda.

Diseño del algoritmo

Inicio
Desmontar la rueda
Desmontar la cubierta
Sacar la cámara

Inflar la cámara
Meter una sección de la cámara en un cubo de agua
Mientras no salgan burbujas
    Meter una sección de la cámara en un cubo de agua
Marcar el pinchazo
Echar pegamento
Mientras no esté seco
    Esperar
Poner el parche
Mientras no esté fijo
    Apretar
Montar la cámara
Montar la cubierta
Montar la rueda

Fin

1.1. Variables, tipos de datos y expresiones

Dato. Es la expresión general que describe los objetos con los cuales opera el algoritmo. El tipo de un dato determina su forma de almacenamiento en memoria y las operaciones que van a poder ser efectuadas con él. En principio hay que tener en cuenta que, prácticamente en cualquier lenguaje y por tanto en cualquier algoritmo, se podrán usar datos de los siguientes tipos:

- Simples
  - Numéricos
    - Entero: Subconjunto finito de los números enteros, cuyo rango o tamaño dependerá del lenguaje en el que posteriormente codifiquemos el algoritmo y de la computadora utilizada.
    - Real: Subconjunto de los números reales limitado no sólo en cuanto al tamaño, sino también en cuanto a la precisión.
  - Lógicos. Son aquellos que solo pueden tener dos valores (verdadero o falso) ya que representan el resultado de una comparación entre otros datos (numéricos o alfanuméricos).
  - Alfanuméricos.
    - Carácter: Conjunto finito y ordenado de los caracteres que la computadora reconoce.
    - Cadena: Los datos (objetos) de este tipo contendrán una serie finita de caracteres, que podrán ser directamente traídos o enviados a/desde la consola.
  - Estructurados
    - Arreglos (vectores, matrices)
    - Registros
    - Archivos

- Punteros

Ejemplos de tipos de datos

Entero: 10, 200, 1500, etc.
Real: 4.5, 2.7, 234.78, etc.
Carácter: “a”, “@”, “”, “1”, etc.
Cadena: “Juan José”, “Esto es una cadena”, “300-4568721”, “Calle 102 Cra. 7ª # 7 - 95”

Constantes. Son datos cuyo valor no cambia durante todo el desarrollo del algoritmo.

Ejemplos:

X ← 100
Y ← 45.56
Nombre ← “Juan José”
Seguir ← V

Variables. Una variable es un objeto cuyo valor puede cambiar durante el desarrollo del algoritmo. Se identifica por su nombre y por su tipo, que podrá ser cualquiera, y es el que determina el conjunto de valores que podrá tomar la variable.

Ejemplos:

Entero: Codigo, días
Real: Nota, sueldo
Cadena: Nombre, Dirección
Carácter: Res, Op
Lógica: Seguir

Expresiones. Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

Ejemplo: a + (b + 3) / c

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

Tipos de operadores.

- *Aritméticos.* Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).
  + Suma
  - Resta
  * Multiplicación
  / División
  ^ o ** Potencia
  Mod Modulo (Residuo de la división entre enteros)

- *Relacionales*. Se utilizan para establecer una relación entre dos o más valores del mismo tipo

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

- *Lógicos.* se utilizan para establecer relaciones entre valores lógicos, pueden ser resultado de una expresión relacional.

  Y ó & And o Y
  O Or u O
  ¬ Not o Negación

Identificadores. Representan los datos de un programa (constantes, variables, tipos de datos). Es una secuencia de caracteres que sirve para identificar una posición en la memoria del computador, que nos permite acceder a su contenido.

Ejemplo:

Nombre
Nota
Sueldo
Código

Funciones.

En los lenguajes de programación existen ciertas funciones predefinidas o internas que aceptan unos argumentos y producen un valor denominado resultado. Como funciones numéricas, se usarán:

Función	Descripción	Tipo de argumento	Resultado
Abs(x)	Valor absoluto de x	Entero o real	Igual que el argumento
Arctan(x)	Arcotangente de x	Entero o real	Real
Cos(x)	Coseno de x	Entero o real	Real
Cuadrado(x)	Cuadrado de x	Entero o real	Igual que el argumento
Ent(x)	Entero de x	Real	Entero
Exp(x)	Elevado a la x	Entero o real	Real
In(x)	Logaritmo neperiano de x	Entero o real	Real
Log10(x)	Logaritmo base 10 de x	Entero o real	Real
Raiz2(x)	Raíz cuadrada de x	Entero de x	Real
Redondeo(x)	Redondea x al entero más próximo	Real	Entero
Seno(x)	Seno de x	Entero o real	Real
Trunc(x)	Parte entera de x	Real	Entero

Las funciones se utilizarán escribiendo su nombre, seguido de loa argumentos adecuados encerrados entre paréntesis, en una expresión.

REPRESENTACIÓN DE ALGORITMOS

Un algoritmo puede ser escrito en castellano narrativo, pero esta descripción suele ser demasiado prolija y, además, ambigua. Para representar un algoritmo se sede utilizar algún método que permita independizar dicho algoritmo de los lenguajes de programación y, al mismo tiempo, conseguir que sea fácilmente codificable.

Los métodos más usados para la representación de algoritmos son:

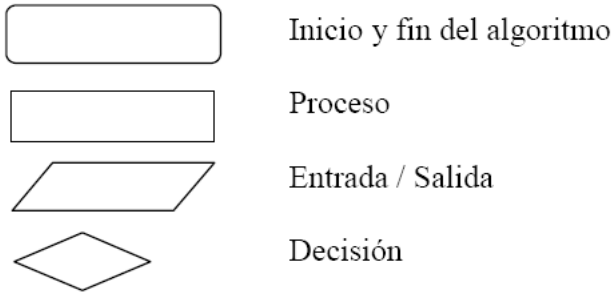
- Diagramas de flujo
- Diagramas N-S (Nassi-Schneiderman)
- Pseudocódigo

Diagramas de flujo. Los diagramas de flujo se utilizan tanto para la representación gráfica de las operaciones ejecutadas sobre los datos a través de todas las partes de un sistema de procesamiento de información, diagramas de flujo del sistema, como para la representación de la secuencia de pasos necesarios para describir un procedimiento particular, diagrama de flujo de detalle.

En la actualidad se siguen usando los diagramas de flujo de sistema, pero ha decaído el uso de los diagramas de flujo de detalle al parecer otros métodos de diseño estructurados más eficaces para la representación y actualización de los algoritmos.

El diagrama de flujo utiliza unos símbolos normalizados, con los pasos del algoritmo escritos en el símbolo adecuado y los símbolos unidos por flechas,

denominadas líneas de flujo, que indican el orden en que los pasos deben ser ejecutados. Los símbolos principales son:



Resulta necesario indicar dentro de los símbolos la operación específica concebida por el programador. Como ejemplo vemos un diagrama de flujo básico, que representa la secuencia de pasos para que un programa lea una temperatura en grados centígrados y calcule y escriba su valor en grados kelvin.

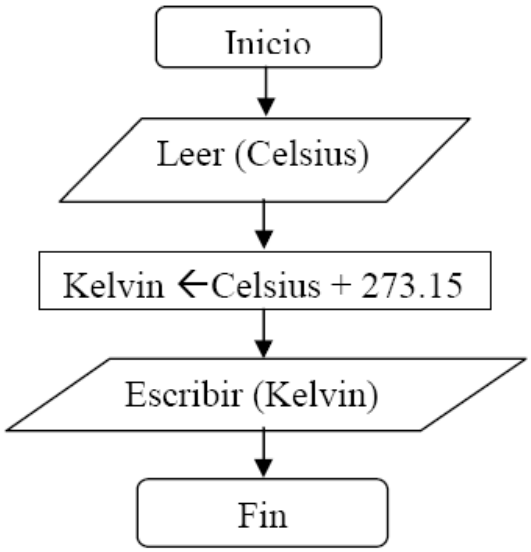
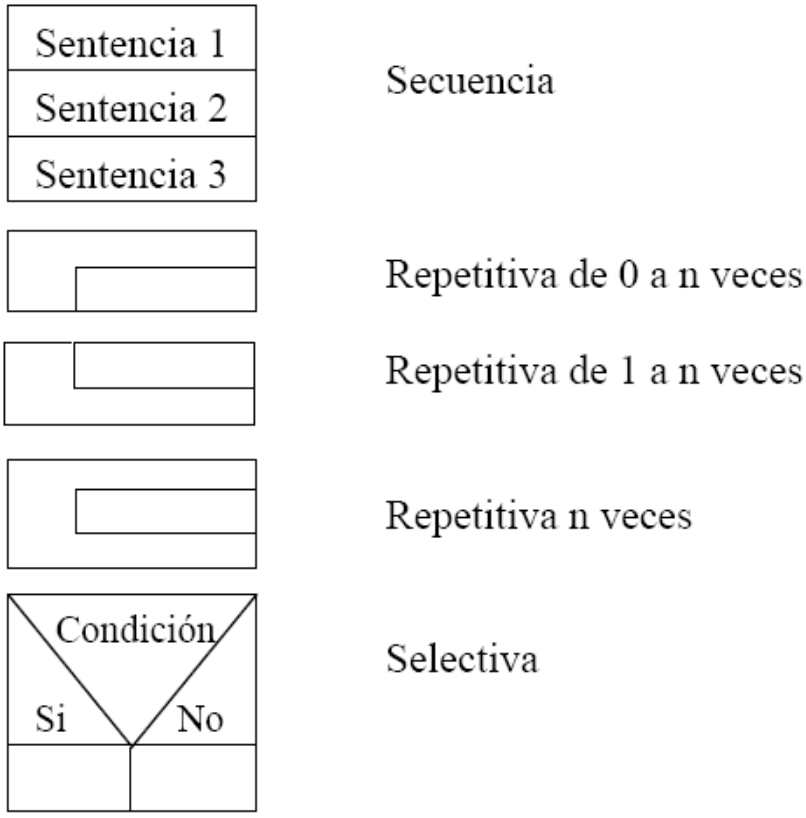


Diagrama Nassi-Schneiderman. Los diagramas N-S denominados así por sus inventores, son una herramienta de programación que favorece la programación estructurada y reúne características gráficas propias de diagrama de flujo y lingüísticas propias de los pseudocódigos. Consta de una serie de cajas continuas que se leerán siempre de arriba-abajo y se documentarán de la forma adecuada.

En los diagramas N-S las tres estructuras básicas de la programación estructurada secuencial, selectiva y repetitiva, encuentran su representación propia. Los símbolos principales son:



El algoritmo que lee una temperatura en grados Celsius y calcula y escribe su valor en grados Kelvin se puede representar mediante el diagrama N-S así:

inicio
leer(Celsius)
Kelvin← Celsius + 273.15
escribir (kelvin)
Fin

Pseudocódigo. El pseudocódigo es el lenguaje de especificación de algoritmos que utiliza palabras reservadas y exige la indentación, o sea sangría en el margen izquierdo de algunas líneas. El pseudocódigo se consiguió para superar las dos principales desventajas del diagrama de flujos: lento de crear y difícil de modificar sin un nuevo redibujo. Es una herramienta muy buena para el seguimiento de la lógica de un algoritmo y para transformar con facilidad los algoritmos a programas, escrito en un lenguaje de programación especifico.

En nuestro pseudocódigo utilizaremos palabras reservadas en español. Así, nuestros algoritmos comenzarán con la palabra reservada ***inicio*** y terminarán con ***fin***, constando de múltiples líneas que se sangran o indentan para mejorar la legibilidad. La estructura básica de un algoritmo escrito en pseudocódigo es:

```
Algoritmo <Identificador_algoritmo>
        // Declaraciones, sentencias no ejecutables
Inicio
        // Acciones, sentencias ejecutables tanto simples como estructuradas
Fin
```

Los espacios en blancos entre los elementos no resultan significativos y las partes importantes se suelen separar unas de otras por líneas en blanco.

Instrucciones de lectura, salida y asignación

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

Leer <variable1>, <variable2>, <variable3>, … <variable$>

Ejemplo:

Leer a, b

Donde “a” y “b” son variables que recibirán valores.

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

Escribir <variable1>, <variable2>, <variable3>, … <variable$>

Ejemplo:

Escribir a, b

También podemos usar un mensaje entre comillas para que el resultado a mostrar o imprimir tenga mayor claridad, ejemplo:

```
Escribir “El valor de A es: ”, a
Escribir “B = ”, b
Escribir “La suma de A+B = ”, a+b
```

Asignación. La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. Se representa con flechas izquierda←, se puede clasificar de la siguiente forma:

- *Simples:* Consiste en pasar un valor constate a una variable, ejemplo a _15
- *Contador:* Consiste en usarla como un verificador del número de veces que se realiza un proceso, ejemplo a ← a + 1
- *Acumulador:* Consiste en usarla como un sumador en un proceso, ejemplo a ← a + b
- *De trabajo:* Donde puede recibir el resultado de una operación matemática que involucre muchas variables, ejemplo a ← (c + b * 2) / 4

Los elementos léxicos de nuestro pseudocódigo son: Comentarios, Palabras reservadas, Identificadores, operadores y literales.

Comentarios. Sirven para documentar el algoritmo y ellos se escriben anotaciones generalmente sobre su funcionamiento. Cuando se coloque un comentario de una sola línea se escribirá precedido de // si el comentario es mistilínea lo pondremos entre {}.

```
//Comentario de una línea
{Comentario que ocupa más
  de una línea}
```

Palabras reservadas. Son palabras que tienen un significado especial como: inicio, fin, leer, escribir, etc.

Identificadores: Son los nombres que se dan a las constantes simbólicas, variables, funciones, procedimientos u otros objetos que manipulan el algoritmo.

La regla para construir un identificador establece que:

- Debe resultar significativo, sugiriendo lo que representa
- No podrá coincidir con las palabras reservadas, propias del lenguaje algorítmico. Como se verá más adelante, la representación de algoritmos mediante pseudocódigo va a requerir la utilización de palabras reservadas.
- Se recomienda un máximo de 50 caracteres.
- Comenzará siempre por un carácter alfabético y los siguientes podrán ser letras, dígitos o el símbolo de subrayado (_)
- Podrá ser utilizado indistintamente escrito en mayúscula o en minúscula

Literales: Los literales son valores que aparecen directamente escritos en el programa y pueden ser literales: lógicos, enteros, reales, de tipo carácter, de tipo cadena y el literal nulo.

EJERCICIOS RESUELTOS

1. ¿Cuál de los siguientes datos son válidos para procesar por una computadora?

a. 3.14159	e. 0.0014	i. 12.5E.3
b. 12345.0	f. 12E+6	j..123E4
c. 15.0E-04	g. 1.1E-3	k. 5A4.14
d. 2.234E2	h. -15E-0.4	l. A1.E04

Serían válidos los datos a, b, c, d, e, f, g, j. Los datos h, i no serían validos pues el exponente no puede tener forma de decimal. El k no sería correcto, pues mezcla caracteres alfabéticos y dígitos y no se puede considerar como un identificador de una variable ya que empieza por un digito. El dato l aunque mescla dígitos y caracteres alfabéticos, podría ser un identificador si admitiéramos el carácter punto como carácter válido para una variable (Pascal o Basic lo consideran).

2. ¿Cuál de los siguientes identificadores son válidos?

a. Renta	e. Dos pulgadas	i. 4A2D2
b. Alquiler	f. C3PO	j. 13Nombre
c. Constante	g. Bienvenido#5	k. Nombres_Apellidos
d. Tom's	h. Elemento	l. NombresApellidos

Se consideran correctos los identificadores a, b, c, f, h, k, l. El d no se considera correcto pues incluye el apóstrofo que no es un carácter válido en un identificador, lo mismo ocurre con el e y el espacio en blanco, y el g con el carácter #. El i, j no serán válidos al no comenzar por un carácter alfabético.

3. Escribir las siguientes expresiones en forma de expresiones algorítmicas

a. $\frac{M}{N} + 4$

b. $\frac{M+N}{P-Q}$

c. $\frac{Seno\ X + Cos\ y}{Tan\ x}$

d. $\frac{-b+\sqrt{b^2-4ac}}{2a}$

Solución

a. (M/N) + 4

b. (M + N) / (P – Q)

c. (Sen(x) + Cos(y)) / Tan(x)

d. (–b + (Raiz2((b ^ 2) – (4 * a * c))) / (2 * a)

4. Escribir un algoritmo que lea un valor entero, lo doble, se multiplique por 10 y visualice el resultado.

Análisis del problema

VARIABLES Y TIPOS
Entero: Número, Resultado
DATOS DE ENTRADA: Numero
DATOS DE SALIDA: Resultado
PROCESO: Resultado ← Numero * 2 * 10

Diseño del algoritmo

Algoritmo Cálculo
Inicio
    Leer Número
    Resultado ← Numero * 2 * 10
    Escribir Resultado
Fin

VALIDACIÓN DEL ALGORITMO O PRUEBA DE ESCRITORIO

Numero = 5 (Leemos el número 5)
Resultado = 5 * 2 * 10 (Asignamos a resultado la multiplicación de 5 * 2 * 10)
Resultado = 100 (Al escribir el resultado se visualiza el valor 100)

Número	Resultado	Salida
5	100	100
2	40	40

Si efectuamos estos mismos cálculos con una calculadora y nos da el mismo resultado, el algoritmo está bien desarrollado, de lo contrario esta malo y se debe revisar y corregir.

5. Diseñar un algoritmo que lea cuatro variables y calcule e imprima su producto, su suma y su media aritmética.

Análisis del problema

VARIABLES Y TIPOS
Entero: a, b, c, d, producto, Suma
Real: Media
DATOS DE ENTRADA: a, b, c, d
DATOS DE SALIDA: Producto, Suma, Media
PROCESO:
Producto ← a * b * c * d

Suma ← a + b + c + d
Media ← Suma / 4

Las variables a, b, c, d, producto y suma podrán ser enteras, pero no así la variable media, ya que la división produce siempre resultados de tipo real.

Diseño del algoritmo
Algoritmo Operaciones
Inicio
    Leer a, b, c, d
    Producto _a * b * c * d
    Suma _ a + b + c + d
    Media _ Suma / 4
    Escribir Producto, Suma, Media
Fin

VALIDACIÓN DEL ALGORITMO O PRUEBA DE ESCRITORIO

a	b	c	d	Producto	Suma	Media	Salida
5	2	1	4	40	12	3	40, 12, 3
3	10	1	0	0	14	3.5	0, 14, 3.5

1.2. Condicionales

Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que en base al resultado de esta comparación, se siga un curso de acción dentro del programa.

Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen tres tipos básicos, las simples y compuestas, condicionales y múltiples.

Condicionales simples y compuestos

Las estructuras condicionales simples se les conocen como “Tomas de decisión”. Estas tomas de decisión tienen la siguiente forma:

Condicionales simples

```
Si <condición> Entonces
        Acción(es)
Fin _ Si
```

Condicionales compuestos

```
Si <condición> Entonces
        Acción(es)
Si no
        Acción(es)
Fin _ Si
```

Condicionales anidados

Decimos que una estructura condicional es anidada cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional:

```
Si <condición1> Entonces
        Si <condición2> Entonces
                …
                Si <condiciónN> Entonces
                        Acción(es)
                Fin _ Si
        Fin _ Si
Fin _ Si
```

Condicionales múltiples

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

```
Si <condición1> Entonces
        Acción(es)
Si no
        Si <condición2> Entonces
                Acción(es)
        Si no
                …
                Si <condiciónN> Entonces
                        Acción(es)
                Si no
                        Acción(es)
                Fin _ Si
```

```
        Fin _ Si
Fin _ Si


Según sea <expresión> Hacer
        <lista1>: Acción1
        <lista2>: Acción2
        …
        <listaN>: AcciónN
[Si no
        AcciónM]
Fin_ según
```

EJERCICIOS RESUELTOS

1. Realizar un algoritmo que lea un número entero y determine si es par o impar.

   Análisis del problema

   VARIABLES Y TIPOS
   Entero: Nro
   DATOS DE ENTRADA: Num

   Diseño del algoritmo

```
Algoritmo Par _ Impar
Inicio
    Escribir "Ingrese un valor entero y positivo"
    Leer Nro
    Si (Nro Mod 2) = 0 Entonces
            Escribir "El valor ingresado es par"
    Si no
            Escribir "El valor ingresado es impar"
    Fin _ si
Fin
```

   VALIDACIÓN DEL ALGORITMO O PRUEBA DE ESCRITORIO

Nro	Salida
10	El valor ingresado es par
11	El valor ingresado es impar

2. Un vendedor recibe un sueldo base más una comisión del 10% si su venta es menor a cien mil pesos o del 15% si su venta es mayor o igual cien mil pesos, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por la venta realizada y el sueldo a devengar.

Análisis del problema

VARIABLES Y TIPOS
Real: SuelBas, Venta, Comi, SuelNeto

DATOS DE ENTRADA: SuelBas, Venta
DATOS DE SALIDA: Comi, SuelNeto
PROCESO: Comi ← Venta * %
SuelNeto ← suelbas + Comi

Diseño del algoritmo

Algoritmo Nomina

Inicio
Escribir "Ingrese el sueldo base"
Leer SuelBas
Escribir "Ingrese el valor vendido"
Leer Venta
Si Venta < 100000 Entonces
Comi ← Venta * 0.10
Si no
Comi ← Venta * 0-15
Fin _ si
SuelNeto ← SuelBas + Comi
Escribir "Comisión $", Comi
Escribir "Sueldo Neto $",SuelNeto
Fin

VALIDACIÓN DEL ALGORITMO O PRUEBA DE ESCRITORIO

SuelBas	Venta	Comi	SuelNeto
200000	130000	19500	149500
220000	80000	8000	228000

1.3. Ciclos

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces.

Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa).

Es frecuente el uso de contadores o banderas para controlar un bucle. También se utilizan con esta finalidad los centinelas.

Un centinela es un valor específico predefinido dado a una variable que permite detectar cuándo se le introduzca, por ejemplo "*", 0, etc.

Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción. Los contadores se utilizan en las estructuras repetitivas con la finalidad de contar sucesos o acciones internas del bucle.

Necesita operaciones de:

- Inicialización
  <Nombre_contador> ← <Valor_de_inicialización>
  C ← 0

- Contador
  <Nombre_contador> ← <Nombre_contador> + <Constante>
  C ← C + 1

La inicialización consiste en asignar al contador un valor. Se situará antes y fuera del bucle.

<Nombre_del_contador> ← <Nombre_del_contador> + <Valor_constante>

Dicho <Valor_constante> podría ser positivo o negativo. Esta instrucción se colocará en el interior del bucle.

C ← C + 1

Un *acumulador* es una variable cuyo valor se incrementa o decrementa en una cantidad determinada. Necesita operaciones de:

- Inicialización
  <Nombre_acumulador> ← <Valor_de_inicialización>
  A ← 0

- Acumulación
  <Nombre_acumulador> _ <Nombre_acumulador> + <Nombre_variable>
  A _ A + Valor

Hay que tener en cuenta que la siguiente también sería una operación de acumulación.

A ← A * Valor

Ciclo Para

Son aquellos en que el número de iteraciones se conoce antes de ejecutarse el ciclo.

La estructura *para* 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 1, o en el valor que especifiquemos, y si este nuevo valor no excede el final se ejecutan de nuevo las acciones.

Sí establecemos que la variable índice se decrementa en cada iteración el valor inicial deberá ser superior al final. Considerando siempre la variable índice de tipo entero.

La forma de esta estructura es la Siguiente:

```
Para Variable _ Valor-inicial Hasta Valor-Final, [incremento | decremento] Hacer
        Acción 1
        Acción 2
        …
        Acción N
Fin _ Para
```

Ciclo Mientras

Las acciones del cuerpo del bucle se repite mientas se cumpla una determinada condición.

Lo que caracteriza este tipo de estructura es que las acciones del cuerpo del bucle se realizan cuando la condición es cierta. Además, se pregunta por la condición al

principio, de donde se deduce que dichas acciones se podrán ejecutar de 0 a n veces.

Mientras <condición> Hacer
    Acción 1
    Acción 2
    …
    Acción N
Fin _ Mientras

Ciclo Repetir

Repite un proceso una cantidad de veces hasta que la condición se cumple. Por otra parte, esta estructura permite realizar el proceso cuando menos una vez, ya que la condición se evalúa al final del proceso.

Repetir
    Acción 1
    Acción 2
    …
    Acción N
Hasta Que <condición>

EJERCICIOS RESUELTOS

1. Calcular el promedio de un alumno que tiene 3 calificaciones en la materia de Algoritmos.

   Análisis del problema

   VARIABLES Y TIPOS
   Real: Nota, SNota, PNota
   Entero: I

   DATOS DE ENTRADA: Nota
   PROCESO: SNota ← SNota + Nota
   PNota = Snota / I
   DATOS DE SALIDA: PNota

   Diseño del algoritmo

   Algoritmo Promedio
   Inicio

```
    SNota ← 0
    Para I ← 1 Hasta 3 Hacer
            Escribir "Ingrese la nota no. ",I
            Leer Nota
            SNota ← SNota + Nota
    Fin _ Para
    PNota ← SNota / I
    Escribir "Nota promedio: ", PNota
Fin
```

VALIDACIÓN DEL ALGORITMO O PRUEBA DE ESCRITORIO

I	Nota	SNota	PNota
1	4.0	4.0	
2	2.5	6.5	
3	3.1	9,6	3.2

2. Realizar un algoritmo que calcule independientemente la suma de los números pares e impares comprendidos entre 1 y 10.

Análisis del problema

VARIABLES Y TIPOS
Entero: Nro, SPar, SImp

PROCESO: SPar _ Spar + Nro
SImp _ SImp + Nro
SALIDA: SPar, SImp

Diseño del algoritmo

```
Algoritmo Sumar
Inicio
    Nro ← 0
    SPar ← 0
    SImp ←0
    Repetir
            Nro ← Nro + 1
            Si Nro mod 2 = 0 Entonces
                    SPar ← SPar + Nro
            Si _no
                    SImp _ SImp + Nro
            Fin _ si
```

```
    Hasta _ que Nro = 10
    Escribir "Sumatoria números pares: ", SPar
    Escribir "Sumatoria números impares: ", SImp
Fin
```

VALIDACIÓN DEL ALGORITMO O PRUEBA DE ESCRITORIO

Nro	SPar	SImp	Salida
0	0	0	--
1	--	1	--
2	2	--	--
3	--	4	--
4	6	--	--
5	--	9	--
6	12	--	--
7	--	16	--
8	20	--	--
9	--	25	--
10	30	--	--
--	--	--	30, 25

Este mismo ejercicio también lo podemos resolver utilizando la sentencia **Para** así:

```
Algoritmo Sumar2
Inicio
    SPar ← 0
    SImp ←0
    Para Nro ← 1 Hasta 10 Hacer
            Si Nro mod 2 = 0 Entonces
                    SPar ← SPar + Nro
            Si _no
                    SImp ← SImp + Nro
            Fin_si
    Fin _ para
    Escribir "Sumatoria números pares: ", SPar
    Escribir "Sumatoria números impares: ", SImp
Fin
```

Nro	SPar	SImp	Salida
--	0	0	--
1	--	1	--
2	2	--	--
3	--	4	--
4	6	--	--
5	--	9	--
6	12	--	--
7	--	16	--
8	20	--	--
9	--	25	--
10	30	--	--
--	--	--	30, 25

También se puede realizar utilizando la sentencia Mientras así:

Diseño del algoritmo

```
Algoritmo Sumar3
Inicio
    Nro ← 0
    SPar ← 0
    SImp ←0
    Mientras Nro < 10 Hacer
            Nro ← Nro + 1
            Si Nro mod 2 = 0 Entonces
                    SPar ← SPar + Nro
            Si _no
                    SImp ← SImp + Nro
            Fin_si
    Fin _ mientras
    Escribir “Sumatoria números pares: ”, SPar
    Escribir “Sumatoria números impares: ”, SImp
Fin
```

VALIDACIÓN DEL ALGORITMO O PRUEBA DE ESCRITORIO

Nro	SPar	SImp	Salida
--	0	0	--
1	--	1	--
2	2	--	--
3	--	4	--
4	6	--	--
5	--	9	--
6	12	--	--
7	--	16	--
8	20	--	--
9	--	25	--
10	30	--	--
--	--	--	30, 25