Instituto de Formación Técnica Superior N° 18
http://www.ifts18.edu.ar

Mansilla 3643 (C1425BBW), Ciudad Autónoma de Buenos Aires

Técnico Superior en Análisis de Sistemas

Algoritmos

Asignatura:
Diagramación Lógica

Autor:
Prof. Leandro E. Colombo Viña

11 de abril de 2016

Índice

1. Concepto de Algoritmo

1.1. ¿Qué es el lenguaje?

Es una serie de símbolos que sirven para transmitir uno o más mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como comunicación. La comunicación es un proceso complejo que requiere una serie de reglas simples, pero indispensables para poderse llevar acabo. Las dos principales son las siguientes:

1. Los mensajes deben correr en un sentido a la vez.
2. Deben forzosamente existir 4 elementos: Emisor, Receptor, Medio de Comunicación y Mensaje.

1.2. Y entonces, ¿qué es un lenguaje de programación?

Podemos decir entonces que es un conjunto de símbolos, caracteres y reglas que le permiten a las personas comunicarse con la computadora. Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, cálculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación. Los lenguajes de programación se clasifican en:

Lenguaje Máquina
Son aquellos cuyas instrucciones son directamente entendibles y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje máquina se expresan en términos de la unidad de información más pequeña el bit (dígito binario 0 o 1).

Lenguaje de Bajo Nivel
En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como mnemónicos para las operaciones y direcciones simbólicas.

Lenguaje de Alto Nivel
Los lenguajes de programación de alto nivel (BASIC, Pascal, Cobol, Fortran, etc.) son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos (en general en inglés), lo que facilita la escritura y comprensión del programa.

1.3. Definición de 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.

- Definición 1: Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.
- Definición 2: Un algoritmo se puede definir como una secuencia finita de instrucciones donde cada una de las cuales tiene un significado claro y puede ser efectuada con una cantidad finita de esfuerzo en una longitud de tiempo también finito.

1.3.1. Características de los Algoritmos

Las características más relevantes de los algoritmos son:

- Finito: Un algoritmo debe siempre terminar después de un número finito de pasos.
- Definido: Cada paso de un algoritmo debe ser definido en forma precisa, estableciendo las acciones que van a efectuar clara y rigurosamente en cada caso.
- Entradas: El algoritmo tiene cero o más entradas, es decir cantidades que se entregan inicialmente al algoritmo antes de su ejecución.
- Salidas: Un algoritmo tiene unas o más salidas, es decir cantidades que tiene una relación especifica respecto a las entradas.
- Efectivo: Generalmente, también se espera que un algoritmo sea efectivo. Esto significa que todas las operaciones ha ser realizadas en el algoritmo deben ser lo suficientemente básicas de modo que puedan en principio ser llevadas a cabo en forma exacta y en un período de tiempo finito por una persona usando lápiz y papel (rutear).

En la práctica, para evaluar un buen algoritmo se considera el tiempo que requiere su ejecución, esto puede ser expresado en términos del número de veces que se ejecuta cada paso. Otros criterios de evaluación pueden ser la adaptabilidad del algoritmo al computador, su simplicidad y elegancia, etc. Algunas veces se tienen varios algoritmos para solucionar el mismo problema, y se debe decidir cual es el mejor. Esto último conduce al "Análisis de Algoritmos". Dado un algoritmo es determinar sus características de desempeño.

1.3.2. Representación de Algoritmos

Consiste en una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso, estos lenguajes algorítmicos pueden ser:

1. Gráficos: Es la representación gráfica de las operaciones que realiza un algoritmo. Principalmente se usan diagramas de flujo o diagramas de Nassi–Schneiderman.
2. No Gráficos: Representa en forma descriptiva las operaciones que debe realizar un algoritmo. Se usa pseudocódigo.

2. Técnicas para la formulación de algoritmos

2.1. Diagramas de Flujo

Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de como deben realizarse los pasos en la computadora para producir resultados. Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre si mediante líneas que indican el orden en que se deben ejecutar los procesos. Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI).

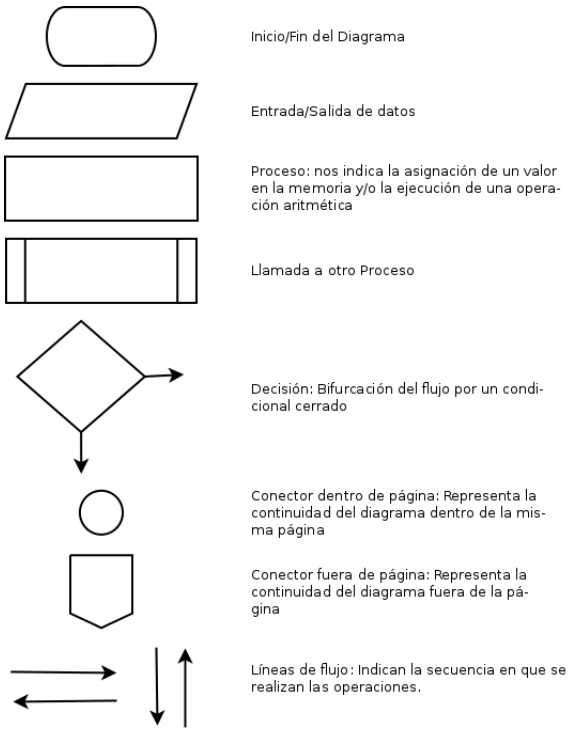

Figura 1: Simbología ANSI para los diagramas de flujo

2.1.1. Reglas para el diseño de Diagramas de Flujo

1. Se deben se usar solamente líneas de flujo horizontales y/o verticales.
2. Se debe evitar el cruce de líneas utilizando los conectores.
3. Se deben usar conectores sólo cuando sea necesario.
4. No deben quedar líneas de flujo sin conectar.
5. Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
6. Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

2.2. Pseudocódigo

Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

2.2.1. Ventajas de utilizar un Pseudocódigo a Diagramas de Flujo

- Ocupa menos espacio en una hoja de papel.
- Permite representar en forma fácil operaciones repetitivas complejas.
- Es muy fácil pasar de pseudocódigo a un programa en algún lenguaje de programación.
- Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

2.3. Diagramas estructurados (Nassi-Schneiderman)

El diagrama estructurado, o diagramas de N-S, muchas veces mal llamado diagrama de Chapin es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas. Las acciones sucesivas se pueden escribir en cajas sucesivas y como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja. Un algoritmo se represente en la sig. forma:

Inicio
Acción 1
Acción 2
...
Acción N
Fin

3. Metodología para Solución de Problemas por Computadora

3.1. Definición del Problema

Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca o entienda del todo no tiene mucho caso continuar con la siguiente etapa.

3.2. Análisis del Problema

Una vez que se ha comprendido lo que se desea del computador, es necesario definir:

- Los datos de entrada.
- Cuál es la información que se desea producir (salida).
- Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy práctica es el que nos pongamos en el lugar de la computadora y analicemos qué es lo que necesitamos que nos ordenen y en que secuencia para producir los resultados esperados.

3.3. Diseño del Algoritmo

Las características de un buen algoritmo son las siguientes:

- Debe poseer un punto particular de inicio.
- Debe ser definido, no debe permitir dobles interpretaciones.
- Debe ser flexible, soportando la mayoría de variantes que se puedan presentar en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.

3.4. Codificación

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas ó en un código reconocible por la computadora. La serie de instrucciones detalladas se le conoce como código fuente, el cuál se escribe en un lenguaje de programación o lenguaje de alto nivel.

3.5. Prueba y Depuración

Los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración. La depuración o prueba resulta una tarea tan creativa como

el mismo desarrollo de la solución, por ello se debe considerar con el mismo interés y entusiasmo.

Resulta conveniente observar los siguientes principios al realizar una depuración, ya que de este trabajo depende el éxito de nuestra solución.

3.6. Documentación

Es la guía o comunicación escrita en sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas.

A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La documentación se divide en tres partes:

Documentación Interna
: Son los comentarios o mensaje que se añaden al código fuente para hacer mas claro el entendimiento de un proceso.

Documentación Externa
: Se define en un documento escrito los siguientes puntos:

  - Descripción del Problema (Enunciado).
  - Nombre del Autor (Analista, Programador).
  - Algoritmo (Diagrama o Pseudocódigo).
  - Diccionario de Datos (Descripción de variables).
  - Código Fuente (Programa).

Manual del Usuario
: Describe paso a paso la manera como funciona el programa, con el fin de que el usuario obtenga el resultado deseado.

3.7. Mantenimiento

Se lleva acabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementación al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

4. Entidades Primitivas para el Desarrollo de Algoritmos

4.1. Tipos de datos

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'b', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.

Tipo de Dato	Simple	Numérico
		Lógico
		Alfanumérico
	Estructurados	Arreglos (Vector y Matrices)
		Registros
		Archivos
		Apuntadores

Es importante destacar que depende del lenguaje de programación que se vaya a utilizar alguno de los tipos de datos que se presentan a continuación puede no existir, o no entrar dentro de la clasificación que aquí se presenta. Tratamos de unificar los diferentes tipos datos desde una perspectiva más global. Al momento de adentrarse en el aprendizaje de cada lenguaje se debe profundizar con qué tipos de datos puede este trabajar.

4.1.1. Tipos de datos simples

- **Datos Numéricos:** Permiten representar valores escalares de forma numérica, esto incluye a los números enteros y los reales. Este tipo de datos permiten realizar operaciones aritméticas comunes.

  `19 -23 3.14`

- **Datos Lógicos:** Son aquellos que sólo pueden tener dos valores (verdadero o falso) ya que representan el resultado de una comparación entre otros datos.

  `True` y `False` o `1` y `0`

- **Datos Alfanuméricos (Cadenas o "String"):** Es una secuencia de caracteres alfanuméricos que permiten representar valores identificables de forma descriptiva, esto incluye nombres de personas, direcciones, etc. Es posible representar números como alfanuméricos, pero estos pierden su propiedad matemática, es decir no se pueden realizar operaciones con ellos. Este tipo de datos por lo general se representan encerrados entre comillas.

  `"Computadora" "Instituto de Formación Técnica Superior"`

4.1.2. Tipos de datos estructurados

- **Arreglos:** son un conjunto finito de valores escalares, ordenados y consecutivos. Un Arreglo es una estructura de datos que almacena bajo el mismo nombre una colección de datos del mismo tipo. Los arreglos se caracterizan por:
  - Almacenan los elementos en posiciones contiguas de memoria.
  - Tienen un mismo nombre de variable que representa a todos los elementos. Para hacer referencia a esos elementos es necesario utilizar un índice que especifica el lugar que ocupa cada elemento dentro del archivo.
  - Si el arreglo es de una única dimensión, es decir, sus elementos se ubican con un único índice, se le llama VECTOR. Si tiene más de una dimensión, osea, más de un índice, se le llama MATRIZ.

VECTOR	
0	dato1
1	dato2
2	dato3
3	dato4
4	dato5
5	dato6
6	dato7

MATRIZ		
	0	1
0	dato01	dato02
1	dato03	dato04
2	dato05	dato06
3	dato07	dato08
4	dato09	dato10
5	dato11	dato12

- **Definidos por el usuario:** Muchos lenguajes de programación nos permiten definir nuestro propios tipos de datos.

4.2. Expresiones

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

```
a + (b + 3)/c
```

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas. Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

- Aritméticas
- Relacionales
- Lógicas

4.3. Operadores

Los operadores son elementos que se relacionan de forma diferente, los valores de una o mas variables y/o constantes. Es decir, los operadores nos permiten manipular valores. Tenemos tres tipos de operadores:

- Aritméticos
- Relacionales
- Lógicos

4.3.1. Operadores Aritméticos

Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes). Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son números enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.

- **Suma**: devuelve la suma entre dos operandos, por lo general se utiliza el símbolo `+`.
- **Resta**: devuelve la resta entre dos operandos, por lo general se utiliza el símbolo `-`.
- **Producto**: devuelve el producto entre dos operandos, por lo general se utiliza el símbolo `*`.
- **Cociente**: devuelve el cociente entre dos operandos, por lo general se utiliza el símbolo `/`.
- **Módulo**: devuelve el resto del cociente entre dos operandos, por lo general se utiliza el símbolo `%` ó `mod`.
- **Potenciación**: devuelve el resultado de elevar el primer operando al segundo, por lo general se utiliza el símbolo `^`. No muchos lenguajes disponen de este operador.

Por ejemplo:

$$7/2 \longrightarrow \mathbf{3} \quad (1)$$
$$7{,}0/2 \longrightarrow \mathbf{3.5} \quad (2)$$
$$12\,\%7 \longrightarrow \mathbf{5} \quad (3)$$
$$4+2*5 \longrightarrow \mathbf{14} \quad (4)$$

4.3.2. Precedencia de los operadores aritméticos

Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el más interno se evalúa primero. Dentro de una misma expresión los operadores se evalúan en el siguiente orden:

1. Potenciación.
2. Producto, cociente y módulo.
3. Suma y resta.

Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha. Es importante destacar que esto puede variar de acuerdo al lenguaje que se esté utilizando. Para ello hay que consultar las caracteristicas del lenguaje en cuestión.

Por ejemplo:

$$4 + 2 * 5 \rightarrow \quad \mathbf{14} \tag{5}$$

$$23 * 2/5 \rightarrow 46/5 \rightarrow \quad \mathbf{9} \tag{6}$$

$$3 + 5 * (10 - (2 + 4)) \rightarrow 3 + 5 * (10 - 6) \rightarrow 3 + 5 * 4 \rightarrow 3 + 20 \rightarrow \quad \mathbf{23} \tag{7}$$

$$3{,}5 + 5{,}09 - 14{,}0/40 \rightarrow 3{,}5 + 5{,}09 - 3{,}5 \rightarrow 8{,}59 - 3{,}5 \rightarrow \quad \mathbf{5.09} \tag{8}$$

$$2{,}1 * (1{,}5 + 3 * 4) \rightarrow 2{,}1 * (1{,}5 + 12) \rightarrow 2{,}1 * 13{,}5 \rightarrow \quad \mathbf{28.35} \tag{9}$$

4.3.3. Operadores Relacionales

Se utilizan para establecer una relación entre dos valores. Compara valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).

Los operadores relacionales comparan valores del mismo tipo, no se tiene sentido realizar comparaciones entre valores que no son del mismo tipo. Por ejemplo querer comparar un número entero con una cadena de texto. Tienen el mismo nivel de prioridad en su evaluación. Y por lo general tienen menor prioridad que los aritméticos.

El valor de verdad que devuelven varía de acuerdo al lenguaje utilizado.

- **Mayor:** se suele utilizar el símbolo `>`.
- **Menor:** se suele utilizar el símbolo `<`.
- **Mayor o igual:** se suele utilizar el símbolo `>=`.
- **Menor o igual:** se suele utilizar el símbolo `<=`.
- **No igual:** se suele utilizar el símbolo `!=` ó `<>`.
- **Igual:** se suele utilizar el símbolo `==`.

Por ejemplo, consideremos $a = 10$, $b = 20$ y $c = 30$:

$$a > c \qquad Falso \tag{10}$$

$$a + b > c \qquad Falso \tag{11}$$

$$a - b <= c \qquad Verdadero \tag{12}$$

$$a + b \ != c \qquad Falso \tag{13}$$

$$a * b == c \qquad Falso \tag{14}$$

4.3.4. Operadores Lógicos

Estos operadores se utilizan para establecer relaciones entre valores lógicos y pueden ser resultado de una expresión relacional.

1. **Suma lógica:** también llamado OR se suele simbolizar con `||`.

Operando 1	Operador	Operando 2	Resultado
Falso	`||`	Falso	**Falso**
Falso	`||`	Verdadero	**Verdadero**
Verdadero	`||`	Falso	**Verdadero**
Verdadero	`||`	Verdadero	**Verdadero**

2. **Producto lógico:** también llamado AND se suele simbolizar con `&&`.

Operando 1	Operador	Operando 2	Resultado
Falso	`&&`	Falso	**Falso**
Falso	`&&`	Verdadero	**Falso**
Verdadero	`&&`	Falso	**Falso**
Verdadero	`&&`	Verdadero	**Verdadero**

3. **Negación:** también llamado NOT se suele simbolizar con `!`, este operador se aplica sobre un único operando.

Operando 1	Operador	Resultado
Falso	`!`	**Verdadero**
Verdadero	`!`	**Falso**

4.3.5. Precedencia de los operadores lógicos

1. Negación.
2. Producto Lógico.
3. Suma Lógica.

4.3.6. Precedencia de los operadores en general

1. Expresiones entre paréntesis: `()`.
2. Potenciación: `^`.
3. Producto, Cociente, Módulo y Negación: `*`, `/`, `%` y `!`.
4. Suma, Resta y Producto lógico: `+`, `-`, `&&`.
5. Operaciones relacionales y Suma lógica: `<`, `<=`, `>`, `>=`, `!=`, `==`, `||`.

Por ejemplo, consideremos $a = 10$, $b = 12$, $c = 3$ y $d = 10$:

$$((a > b + c)||(a < c\hat{}2))\&\&((a == c)||(a <= b * d)) \quad (15)$$
$$((a > b + c)||(a < 9))\&\&((a == c)||(a <= b * d)) \quad (16)$$
$$((a > b + c)||(a < 9))\&\&((a == c)||(a <= 120)) \quad (17)$$
$$((a > 15)||(a < 9))\&\&((a == c)||(a <= 120)) \quad (18)$$
$$((10 > 15)||(10 < 9))\&\&((10 == 3)||(10 <= 120)) \quad (19)$$
$$(Falso||Falso)\&\&(Falso||Verdadero) \quad (20)$$
$$Falso\&\&Verdadero \quad (21)$$
$$\textbf{Verdadero} \quad (22)$$

4.4. Identificadores

Los identificadores representan los datos de un programa (constantes, variables, tipos de datos). Un identificador es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, que nos permite accesar a su contenido.

Esta secuencia es un nombre elegido por el programador ya que es mucho más fácil recordar un nombre que una posición de memoria. Como por ejemplo:

```
Nombre Num_hrs Cantidad
```

4.4.1. Reglas para formar un identicador

Estas reglas pueden variar acorde a las características sintácticas del lenguaje en el que estés programando. Pero pueden nombrarse algunas buenas prácticas que se respetan en la mayoría de los lenguajes.

- Debe comenzar con una letra (A a Z, mayúsculas o minúsculas) y no deben contener espacios en blanco.
- Letras, dígitos y caracteres como el guión bajo (_) están permitidos después del primer carácter. En algunos lenguajes estas prácticas tiene significados particulares.
- La longitud de identificadores no debe ser demasiado larga. Pero debe alcanzar para ser descriptor de su contenido.
- Los identificadores que se escriben todos con mayúscula por lo general son constantes.

4.4.2. Constantes y Variables

Una **constante** es un dato numérico o alfanumérico que no cambia durante la ejecución del programa. Ejemplo:

```
pi = 3,1416
```
(23)

En cambio una **variable** es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo. Ejemplo:

```
perimetro = 2 * pi * radio
```
(24)

Considerando 24 y 23, las `variables` son `radio`, `perimetro` y las constates son `pi` y 2.

4.4.3. Clasificación de variables

Variables	Por su contenido	Numéricas
		Lógicas
		Alfanuméricas
	Por su uso	De trabajo
		Contadores
		Banderas
		Acumuladores

- Por su contenido:
  - **Variables Numéricas:** Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal. Por ejemplo:

    ```
    iva=0,21 pi=3,1416 costo=2500
    ```

  - **Variables Lógicas:** Son aquellas que sólo pueden tener dos valores (verdadero o falso) estos representan el resultado de una comparación entre otros datos.
  - **Variables Alfanuméricas:** Esta formada por caracteres alfanuméricos (letras, números y caracteres especiales). Por ejemplo:

    ```
    letra='a' apellido='lopez' direccion='Mansilla 3643'
    ```

- Por su uso:
  - **Variables de Trabajo:** reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa. Por ejemplo:

    ```
    suma=a+b/c
    ```

  - **Contadores:** Se utilizan para llevar el control del número de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno. Por ejemplo:

    ```
    i=i+1
    ```

  - **Acumuladores:** Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente. Por ejemplo:

    ```
    Sumador = Sumador + variable
    ```

5. Estructuras Algorítmicas

Las estructuras de operación de programas son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución deproblemas. Estas estructuras se clasifican de acuerdo con su complejidad en:

Estructuras Algorítmicas	Secuenciales	Asignación
		Entrada
		Salida
	Condicionales	Simples
		Múltiples
	Cíclicas	Hacer para
		Mientras
		Hacer mientras

5.1. Estructuras secuenciales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. Una estructura secuencial se representa de la siguiente forma:

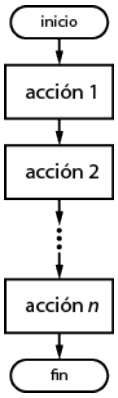

Figura 2: Diagrama de Flujo de estructuras secuenciales

Dentro de las estrucuras secuenciales podemos encontrar:

- Asignación
- Lectura
- Escritura

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. La asignación se puede clasificar de la siguientes formas:

- **Simples:** Consiste en pasar un valor constate a una variable, se utiliza el símbolo `=`. Por ejemplo: `a = 15`.
- **Contador:** Consiste en usarla como un verificador del numero de veces que se realiza un proceso. Por ejemplo: `a = a + 1`.

- **Acumulador:** Consiste en usarla como un sumador en un proceso. Por ejemplo: `a = a + b`
- **De trabajo:** Donde puede recibir el resultado de una operación matemática que involucre muchas variables. Por ejemplo: `a = c + b * 2`

La operación de asignación se representa con un rectángulo que contiene la expresión de asignación.

La **lectura** consiste en recibir desde un dispositivo de entrada (p.ej. el teclado) un valor. Esta operación se representa con un paralelogramo que contiene el nombre de la variable donde se almacena el valor leído con una flecha que ingresa al paralelogramo.

La **escritura** consiste en mandar por un dispositivo de salida (p.ej. monitor o impresora) un resultado o mensaje. Este proceso se representa con un paralelogramo que contiene el nombre de la variable cuyo valor se escribe con una flecha que sale del paralelogramo.

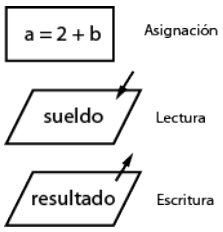

Figura 3: Símbolos de asignación, lectura y escritura