

CAPITULO : 2

LA RESOLUCION DE PROBLEMAS CON COMPUTADORAS Y LAS HERRAMIENTAS DE PROGRAMACIÓN

Contenido:

- 2.1.-La resolución de problemas
- 2.2.-Análisis del problema.
- 2.3.-Diseño del algoritmo
- 2.4.-Resolución del problema mediante computadora
- 2.5.-Representación gráfica de los algoritmos
- 2.6.-Diagramas Nassi Schneiderman. (N-S)
- 2.7.-Pseudocódigo.

ACTIVIDADES DE PROGRAMACIÓN RESUELTAS EJERCICIOS :

La resolución de problemas con computadora se puede resolver en tres fases:

- ↔ *análisis del problema*
- ↔ *diseño del algoritmo*
- ↔ *resolución del algoritmo en la computadora.*

En este capítulo se analizan las tres fases anteriores. El análisis y el diseño del algoritmo requieren la descripción del problema en subproblemas a base de "refinamientos sucesivos" y una herramienta de programación -diagramas de flujo, diagrama NS o pseudocódigo- ; los conceptos fundamentales del análisis, diseño y herramientas de programación (diagramas de flujo, diagramas NS y pseudocodigos) se describen como conocimientos indispensables para el aprendizaje de la programación de computadoras.

2.1.- LA RESOLUCIÓN DE PROBLEMAS.

La principal razón para que las personas aprendan a programar en general y los lenguajes de programación en particular es utilizar la computadora como una herramienta para la resolución de problemas. Ayudado por una computadora, la resolución de un problema se puede dividir en tres fases importantes:

- 1.-Análisis del problema.
- 2.-Diseño o desarrollo del algoritmo.
- 3.-Resolución del algoritmo en la computadora.

El primer paso –Análisis del problema- requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle. Una vez analizado el problema, se debe desarrollar el algoritmo –procedimiento paso a paso para solucionar el problema dado-. Por último, para resolver el algoritmo mediante una computadora, se necesita codificar el algoritmo en un lenguaje de programación Pascal, C/++, Cobol, Fortran, etc. , es decir, convertir el algoritmo en programa, ejecutarlo y comprobar que el programa soluciona verdaderamente el problema. Las fases del proceso de resolución de un problema mediante computadora se indican en la figura 2.1.

2.1.-ANÁLISIS DEL PROBLEMA.

El propósito del análisis de un problema, es ayudar al programador para llegar a una cierta comprensión de la naturaleza del problema. El problema debe estar bien definido si ase desea llegar a una solución satisfactoria.

Para poder definir con precisión el problema se requiere que las especificaciones de entrada y salida sean descritas con detalle. Una buena definición del problema, junto con una descripción detallada de las especificaciones de entrada y salida, son los requisitos más importantes para llegar a una solución eficaz.

El análisis del problema exige una lectura previa del problema a fin de obtener una idea general de lo que se solicita. La segunda lectura deberá servir para resolver a las preguntas:

↔ ¿Qué información debe proporcionar la resolución del problema?

↔ ¿Qué datos se necesitan para resolver el problema?

La respuesta a la primera pregunta indicará los resultados deseados a las *salidas del problema*.

La respuesta a la segunda pregunta indicará que datos se proporcionan a las *entradas del problema*.

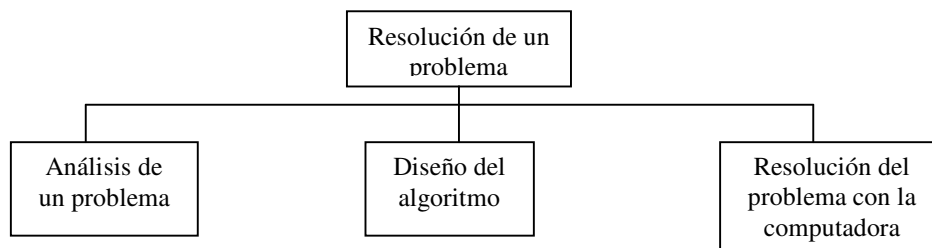


Figura 2.1 la resolución de un problema

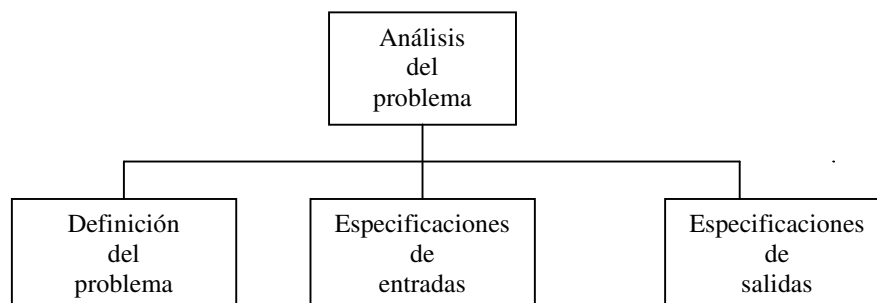


Figura 2.2 Análisis del problema

Ejemplo 2.1.

Leer el radio de un círculo y calcular e imprimir su superficie y la longitud de la circunferencia.

Análisis.

Las entradas de datos en este problema se concentran en el radio del círculo. Dado que el radio puede tomar cualquier valor dentro del rango de los números reales, el tipo de datos radio debe ser real.

Las salidas serán dos variables: superficie y circunferencia, que también serán de tipo real.

Entradas: radio del círculo(variable RADIO).
Salidas: superficie del círculo(variable Area).
Circunferencia del círculo(variable Circunferencia).
Variables: Radio, Área y circunferencia (tipo real).

2.3.-DISEÑO DEL ALGORITMO.

Una computadora no tiene capacidad para solucionar problemas mas que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutar por la maquina, constituyen, como ya conocemos, *el algoritmo*.

La información proporcionada al algoritmo, constituye su *entrada* y la información producida por el algoritmo constituye su *salida*.

Los problemas complejos se pueden resolver mas eficazmente con la computadora, cuando se rompen en sub problemas que sean más fáciles de solucionar que el original. Este método se suele denominar *divide y vencerás (divide and conquer)* que consiste en dividir un problema complejo en otros más simples. Así el problema de encontrar la superficie y longitud de un círculo se puede dividir en tres problemas más simples o *subproblemas* (figura 2.3.)

La descomposición del problema original en subproblemas más simples y a continuación dividir estos subproblemas en otros más simples que pueden ser implementados para su solución en la computadora se denomina *diseño descendente (top-down design)*. Normalmente los pasos diseñados en el primer esbozo del algoritmo son incompletos e indicaran solo unos pocos pasos (un máximo de doce aproximadamente). Tras esta primera descripción, estos se amplían en una descripción mas detallada con más pasos específicos. Este proceso se denomina *financiamiento del algoritmo (stepwise refinement)*. Para problemas complejos se necesitan con frecuencia diferentes *niveles de refinamiento* antes de que se pueda obtener un algoritmo claro, preciso y completo.

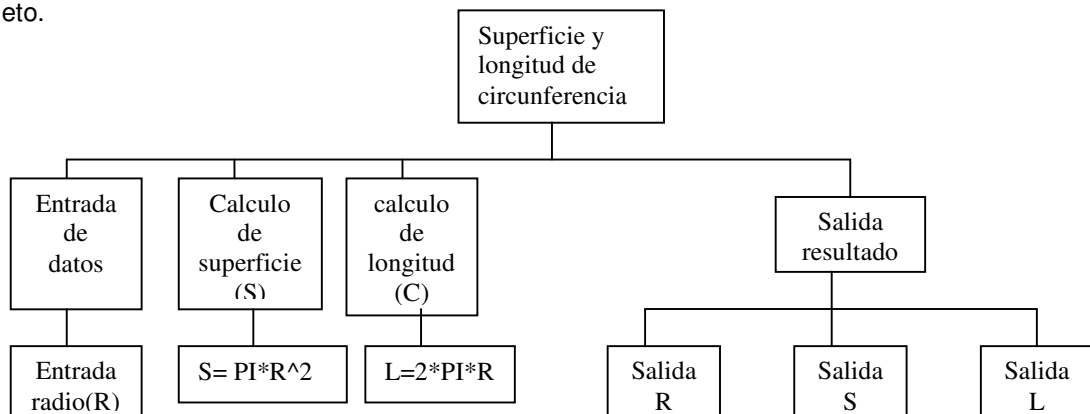


Figura 2.3. refinamiento de un algoritmo

El problema de calculo de la circunferencia y superficie de un circulo se puede descomponer en subproblemas más simples: (1) leer datos de entrada, (2) calcular superficie y longitud de circunferencia y (3) escribir resultados (datos de salida).

Subproblema	Refinamiento
<i>leer</i> radio calcular superficie calcular circunferencia <i>escribir</i> resultados	leer radio superficie= $3.141592 * \text{radio}^2$ circunferencia $2 * 3.141592 * \text{radio}$ <i>escribir</i> radio, circunferencia, superficie

Las ventajas más importantes del diseño descendente son:

- ↔ El problema se comprende más fácilmente al dividirse en partes más simples denominadas *módulos*.
- ↔ Las modificaciones en los módulos son más fáciles.
- ↔ La comprobación del problema se puede verificar fácilmente.

Tras los pasos anteriores (*diseño descendente y refinamiento por pasos*) es preciso representar el algoritmo mediante una determinada herramienta de programación: diagrama de flujo, pseudocódigo o diagrama N.S

Así pues, el diseño de algoritmo se descompone en las fases recogidas en la Figura 2.4.

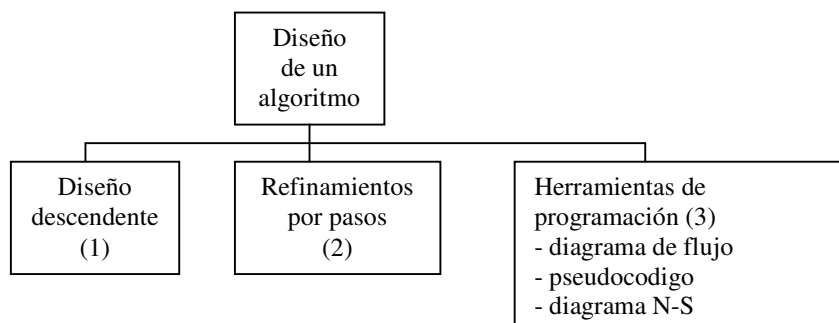


FIGURA 2.4 FASES DEL DISEÑO DE UN ALGORITMO

2.3.1 Escritura inicial del algoritmo.

Como ya se ha comentado anteriormente, el sistema para describir ("escribir") un algoritmo consiste en realizar una descripción paso a paso con un lenguaje natural del citado algoritmo. Recordemos que un algoritmo es un método o un conjunto de reglas para solucionar un problema. En cálculos elementales estas reglas tienen las siguientes propiedades:

- λ deben de estar seguidas de algunas secuencias definidas de pasos hasta que se obtenga un resultado coherente,
- λ Sólo puede ejecutarse una operación a la vez.

El flujo de control usual de un algoritmo es secuencial; consideremos el algoritmo que responde a la pregunta:

¿Qué hacer para ver la película Tiburón?

La respuesta es muy sencilla y puede ser descrita en forma de algoritmo, general de modo similar a:

```
ir al cine
comprar una entrada (billete o ticket)
ver la película
regresar a casa
```

El algoritmo consta de cuatro acciones básicas, cada una de las cuales debe ser ejecutada antes de realizar la siguiente. En términos de computadora, cada acción se codificará en una o varias sentencias que ejecutan una tarea particular.

El algoritmo descrito es muy sencillo; sin embargo, como ya se ha indicado en párrafos anteriores, el algoritmo general se descompondrá en pasos más simples en un procedimiento denominado *refinamiento sucesivo*, ya que cada acción puede descomponerse a su vez en otras acciones simples.

Así, un primer refinamiento del algoritmo *ir al cine* se puede describir de la forma siguiente:

```
1. Inicio
2. Ver la cartelera de cines en el periódico
3. Si no proyectan "Tiburón" entonces
    3.1 decidir otra actividad
    3.2 bifurcar al paso 7
    si_no
    3.3 ir al cine.
    Fin_si
4. Si hay cola
    4.1 ponerse en ella
    4.2 mientras haya personas delante hacer
        4.2.1 avanzar en la cola
    fin_mientras
    fin_si
5. Si hay localidades entonces
    5.1 comprar una entrada
    5.2 pasar a la sala.
    5.3 localizar la(s) butaca(s)
    5.4 mientras proyectan la película hacer
        5.4.1 ver la película
    fin_mientras
    5.5 abandonar el cine
    si_no
    5.6 refunfuñar
    fin_si
6. volver a casa
7. fin
```

En el algoritmo anterior existen diferentes aspectos a considerar. En primer lugar, ciertas palabras reservadas se han escrito deliberadamente en negrita (**mientras**, **si no**, etc.). Estas palabras describen las estructuras de control fundamentales y procesos de toma de decisión en el algoritmo. Estas incluyen los conceptos importantes de decisión de selección (expresadas por **si-entonces-si_no** if-then-else) y de repetición (expresadas con **mientras-hacer** o a veces **repetir-hasta e iterar-fin-iterar**, en inglés , while-do y repeat-until) que se encuentra en casi todos los algoritmos , especialmente los de proceso de datos . La capacidad de decisión permite seleccionar alternativas de acciones a seguir o bien la repetición una y otra vez de operaciones básicas

```
Si proyectan la película seleccionada ir al cine
si_no ver la televisión , ir al fútbol o leer el periódico
```

mientras haya personas en la cola , ir avanzando repetidamente hasta llegar a la taquilla.

Otro aspecto a considerar es el método elegido para describir los algoritmos : empleo de indentación (sangrado o justificación) en escritura de algoritmos. En la actualidad es tan importante la escritura de programa como su posterior lectura . Ello se facilita con la indentación de las acciones interiores a las estructuras fundamentales citadas : selectivas y repetitivas . A lo largo de todo el libro la indentación o sangrado de los algoritmos será norma constante.

Para terminar estas consideraciones iniciales sobre algoritmos , describiremos las acciones necesarias para refinar el algoritmo objeto de nuestro estudio ; ello analicemos la acción.

Localizar las butaca(s)

Si los números de los asientos están impresos en la entrada la acción compuesta se resuelve con el siguiente algoritmo:

- 1.**inicio** //algoritmo para encontrar la butaca del espectador
- 2.caminar hasta llegar a la primera fila e butaca
- 3.**repetir**
 - compara numero de fila con numero impreso en billete
 - si** no son iguales , **entonces** pasar a la siguiente fila
 - hasta que** se localice la fila correcta
- 4.**mientras** número de butaca no coincida con número de billete
 - hacer** avanzar a través de la fila a la siguiente butaca
 - fin_mientras**
- 5.sentarse en la butaca
- 6.**fin**

En este algoritmo la repetición se ha mostrado de los modos , utilizando ambas notaciones , **repetir...hasta que** y **mientras...fin_mientras**. Se ha considerado también , como ocurre normalmente , que le número del asiento y fila coincide con el número y fila rotulado en el billete

2.4.RESOLUCIÓN DEL PROBLEMA MEDIANTE COMPUTADORA

Una vez que el algoritmo está diseñado y representado gráficamente mediante una herramienta de programación (diagrama de flujo , pseudocódigo o diagrama N-S) se debe pasar a la fase de resolución práctica del problema con la computadora .

Esta fase se descompone a u vez en las siguientes subfase:

1. codificación del algoritmo en un programa .
2. ejecución del programa .
3. comprobación del programa .

En el diseño del algoritmo éste describe en una herramienta de programación tal como un diagrama de flujo , diagrama N-S o pseudocódigo .Sin embargo , el programa que implementa el algoritmo debe ser escrito en un lenguaje de programación y siguiendo las reglas gramaticales o sintaxis del mismo .La fase de conversión del algoritmo en un lenguaje

de programación se denomina codificación , ya que el algoritmo escrito en un lenguaje específico de programación se denomina código.

Tras la codificación del programa , deberá ejecutarse en una computadora y a continuación de comprobar los resultados pasar a la fase final de documentación.

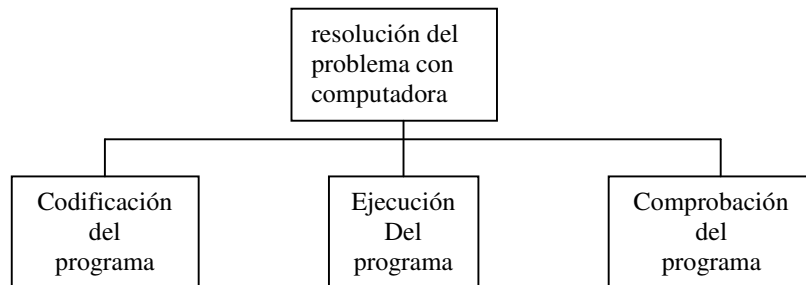


Figura 2.5 resolución del problema mediante computadora

2.5.REPRESENTACIÓN GRAFICA DE LOS ALGORITMOS

Para representar un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Ello permitirá que un algoritmo pueda ser codificado indistintamente en cualquier lenguaje . Para conseguir este objetivo se precisa que el algoritmo sea representado gráfica o numéricamente , de modo que las sucesivas acciones no dependan de la sintaxis de ningún lenguaje de programación , sino que la descripción pueda servir fácilmente para su transformación en un programa , es decir ,su codificación.

Los métodos usuales para representar un algoritmo son:

1. diagrama de flujo,
2. diagrama N-S(Nassi-Schneiderman),
3. lenguaje de especificación de algoritmos :pseudocódigo,
4. lenguaje español,
5. fórmulas .

Los métodos 4 y 5 no suelen ser fáciles de transformar en programas. Una descripción en español narrativo no es satisfactoria , ya que es demasiado prolija y generalmente ambigua. Una fórmula , sin embargo , es buen sistema de representación . Por ejemplo , la fórmula para la solución de una ecuación cuadrática es un medio sucinto de expresar el procedimiento algoritmo que se debe ejecutar para obtener las raíces .

$$X1 = (-b + \sqrt{b^2 - 4ac}) / 2a$$

$$X2 = (-b - \sqrt{b^2 - 4ac}) / 2a$$

Significa

1. Eleve al cuadrado b.
2. Toma a ; multiplicar por 4.
3. Restar el resultado de 2 del resultado de 1 , etc.

Sin embargo , no es frecuente que un algoritmo pueda ser expresado por medio de una simple fórmula .



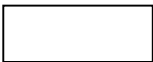
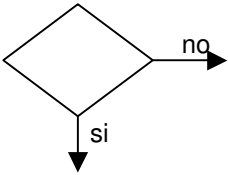
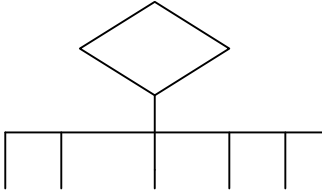

2.5.1. DIAGRAMAS DE FLUJO

Un **diagrama de flujo** (flowchart) es una de las técnicas de representación de algoritmo más antigua y a la vez más utilizada , aunque se empleo ha disminuido considerablemente , sobre todo desde la aparición de lenguajes de programación estructurados . Un diagrama de flujo es un diagrama que utiliza los símbolos (cajas) estándar mostrados en la figura 2.6 y que tiene los pasos del algoritmo escritos en esas cajas unidas por flechas , denominadas líneas de flujo, que indican la secuencia en que se deben ejecutar.

La figura 2.7 es un diagrama de flujo básico.

El diagrama citado (figura 2.7) representa la resolución de un programa que deduce el salario neto de un trabajador a partir de la lectura del nombre , horas trabajadas , precio de la hora , y sabiendo que los impuestos aplicados son el 25 por 100 sobre el salario bruto.

Los símbolos estándar normalizados por ANSI (abreviatura de America National Standard Institute) son muy variados . En la figura 2.8 se representan una plantilla de dibujo típica donde se

Símbolos Principales	Función
	Terminal (representa el comienzo, "inicio" y el final, "fin", de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar en un programa).
	Entrada/Salida (cualquier tipo de introducción de datos en la memoria desde los periféricos "entrada", o registro de la información procesada en un periférico, "salida").
	Proceso (cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etc.).
	Decisión (indica operaciones lógicas o de comparación entre datos –normalmente dos- y en función del resultado de la misma determina cual de los distintos caminos alternativos de programa se debe seguir; normalmente tiene dos salidas –respuestas SI o NO-, pero puede tener tres o mas, según los casos).
	Decisión múltiple (en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).
	Conector (sirve para enlazar dos partes cualesquiera de un ordinograma a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma pagina del diagrama).

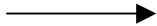



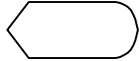


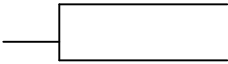
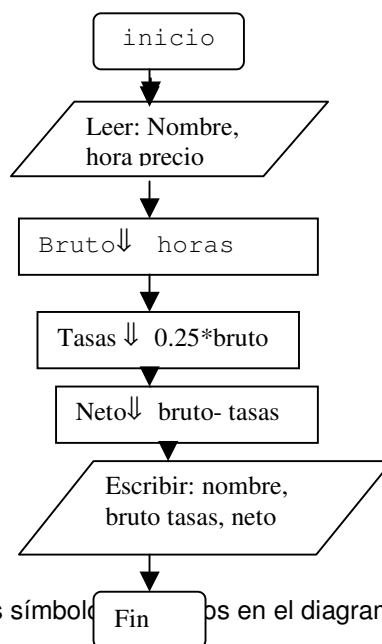
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones).
	Línea conectora (sirve de unión entre dos símbolos).
	Conector (conexión entre dos puntos del organigrama situado en paginas diferentes).
	Llamada a subrutina o a un proceso predeterminado (una subrutina es un modulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa, al terminar, al programa principal).
Símbolos Secundarios	Función
	Pantalla (se utiliza en ocasiones, en lugar del símbolo de E/S).
	Impresora (se utiliza en ocasiones en lugar del símbolo DE/S).
	Teclado (se utiliza en ocasiones en lugar del símbolo de E/S).
	Comentarios (se utiliza para añadir comentarios clasificadores a otro símbolos del diagrama de flujo. Se puede dibujar a cualquier lado del símbolo)

Figura 2.6 símbolos del diagrama de flujo



Contemplan la mayoría de los símbolos en el diagrama ; sin embargo, los símbo_

los más utilizados representan:

- λ Proceso ,
- λ Decisión,
- λ Conectores,
- λ Fin ,
- λ Entrada / salida,
- λ Dirección del flujo,

Y se resumen en la figura 2.9
En un diagrama de flujo:

- λ existe una caja etiquetada "inicio", que es de tipo elíptico .,
- λ existe otra caja etiquetada "fin" de igual forma que la anterior.
- λ Si existen otras cajas, normalmente son regulares, tipo rombo o paralelogramo (el resto de las figuras se utilizan solo en diagramas de flujo generales o de detalle y no siempre son impredecibles).

Se puede escribir mas de un paso del algoritmo en una sola caja rectangular. El uso de flechas significa que la caja no necesita ser escrita debajo de su predecesora. Sin embargo, abusar demasiado de esta flexibilidad conduce a diagramas de flujo complicados e ininteligibles.

Falta

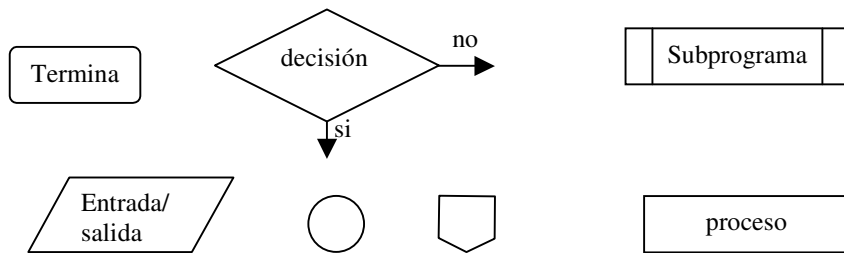


Figura 2.9 símbolos utilizados en los diagramas de flujos

Símbolos de diagramas de flujo

Cada símbolo visto anteriormente indica *el tipo de operación a ejecutar* y el diagrama de flujo ilustra gráficamente la secuencia en la que *se ejecutan las operaciones*.

Las *líneas de flujo* (\diamond) representa el flujo secuencial de la lógica del problema.

Un rectángulo (\square) significa algún tipo de *proceso en la computadora*, es decir, acciones a realizar (sumar dos números, calcular la raíz cuadrada de un número, etc.).

El paralelogramo (\parallel) es un símbolo de entrada / salida que representa cualquier tipo de entrada o salida desde el programa o sistema; por ejemplo, entrada de teclado, salida en impresora o pantalla, etc.







El símbolo rombo (\diamond) es una caja de decisión que representa respuestas sí/no o bien diferentes alternativas 1, 2, 3, 4, ...etc.

Cada diagrama de flujo comienza y termina con un símbolo terminal (\square).

Un pequeño círculo (\circ) es un conector y se utiliza para conectar caminos, tras roturas previas del flujo del algoritmo.

Otros símbolos de diagramas de flujo menos utilizados de mayor detalle que los anteriores son:

Un trapecoide (∇) indica que un proceso manual se va a ejecutar en contraste con el rectángulo, que indica proceso automático.

El símbolo general de entrada/salida se puede subdividir en otros símbolos: teclado (); pantalla (), impresora (), disco magnético (), disquete o disco flexible (), **casete** ().

Ejemplo 2.2

Calcular la media de una serie de números positivos, suponiendo que los datos se leen desde un terminal. Un valor de cero – como entrada – indicara que se ah alcanzado el final de la serie de números positivos.

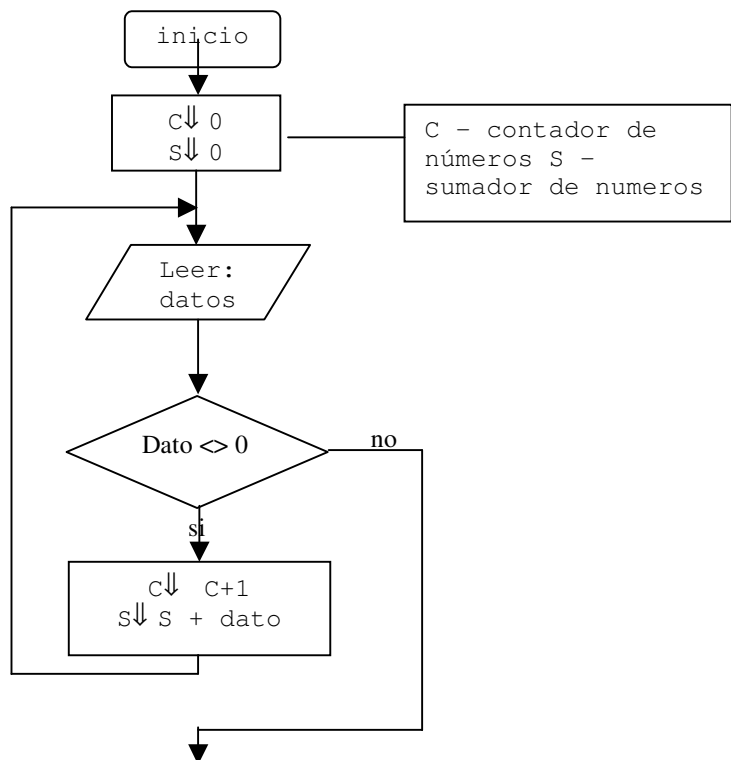
El primer paso a dar en el desarrollo del algoritmo es descomponer el problema en una serie de pasos secuenciales. Para calcular una media se necesita sumar y contar los valores.

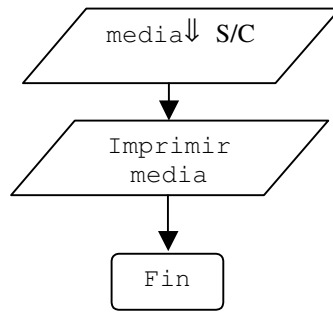
Por consiguiente, nuestro algoritmo en forma descriptiva sería:

1. Inicializar contador de números C y variable suma S.
2. Leer un número.
3. Si el número leído es cero:
 - λ Calcular la media;
 - λ Imprimir la media;
 - λ Fin del proceso**Si** el número leído no es cero:
 - λ Calcular la suma;
 - λ Incrementar en uno el contador de números ;
 - λ Ir al paso 2.
4. **Fin.**

El refinamiento del algoritmo conduce a los pasos sucesivos para realizar las operaciones de lectura de datos, verificación del último dato, suma y media de los datos.

Diagrama de flujo 2.2



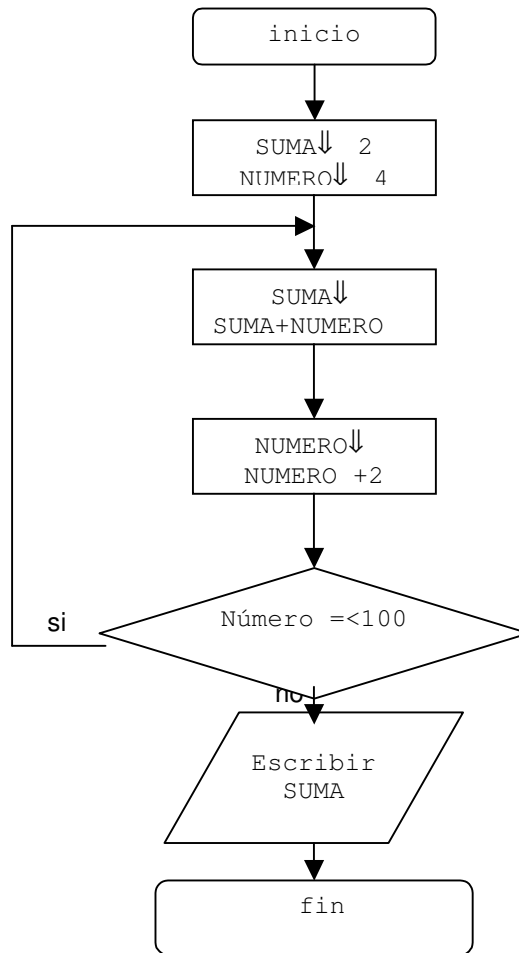


Si el primer dato leído es 0, la división S/C producirá un error si se ejecutara el algoritmo en una computadora, ya que no está permitida en ella la división por cero

Si el primer dato leído es 0, la división S/C produciría un error si se ejecutara el algoritmo en una computadora, ya que no está permitida en ella la división por cero.

Ejemplo 2.3

Suma de los números pares comprendidos entre 2 y 100
Diagrama de flujo 2.3



Ejemplo 2.4

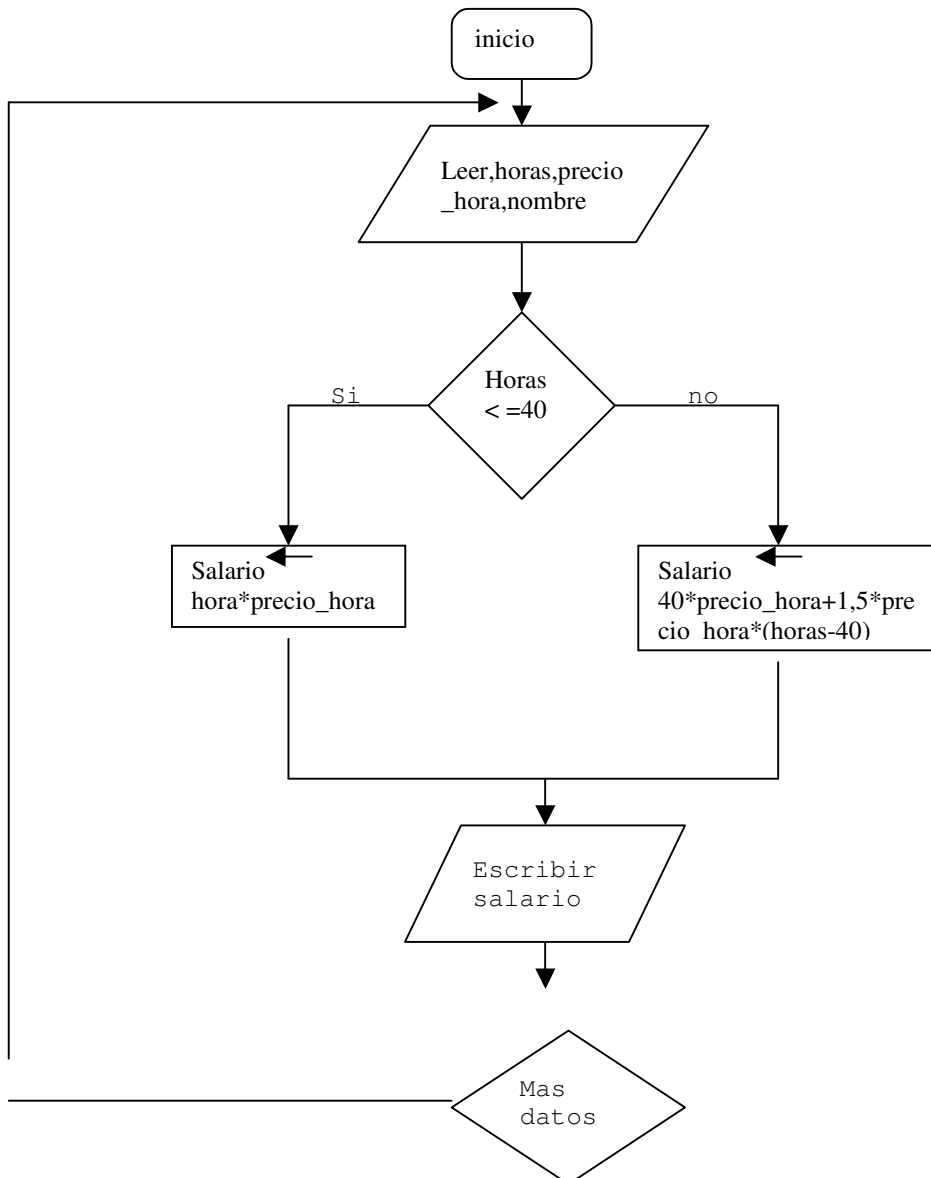
Se desea realizar el algoritmo que resuelva el sistema siguiente:
Cálculo de los salarios mensuales de los empleados de una empresa, sabiendo que éstos se calculan en base a las horas semanales trabajadas y de acuerdo a un precio especificado por hora. Si se pasan de cuarenta horas semanales, las horas extraordinarias se pagarán a razón de 1.5 veces la hora ordinaria

Los cálculos son:

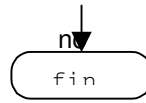
1. Leer datos del archivo de la empresa, hasta que se encuentre la ficha final del archivo (HORAS, PRECIO_HORA, NOMBRE).
2. Si $horas \leq 40$, entonces SALARIO es el producto de horas por PRECIO_HORA.
3. Si $HORA > 40$, entonces salario es la suma de 40 veces PRECIO_HORA más 1.5 veces PRECIO_HORA por $(HORAS=40)$.

El diagrama de flujo completo del algoritmo se indica a continuación:

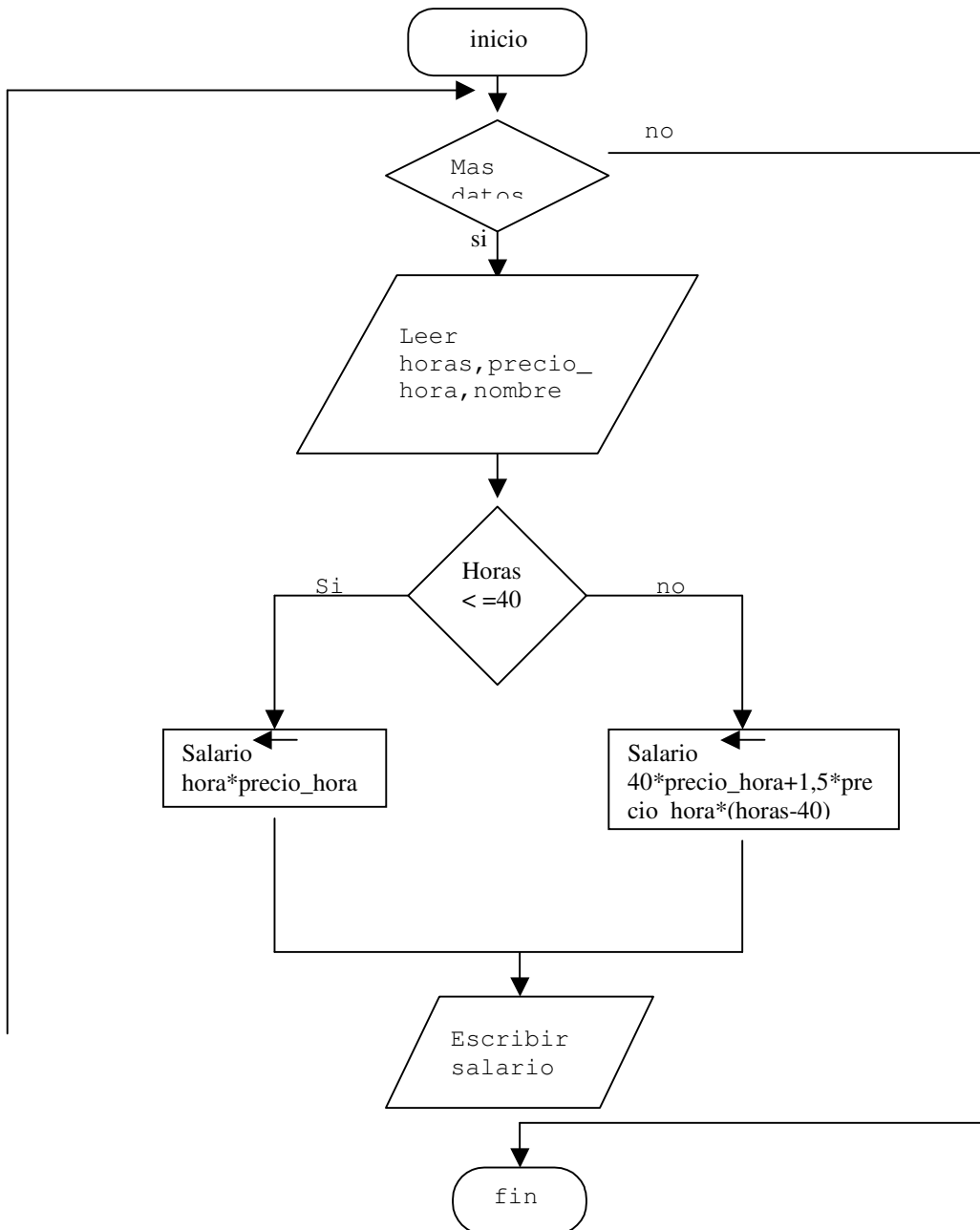
Diagrama de flujo 2.4.a



si



O bien.
Diagrama de flujo 2.4.b



Ejemplo 2.5

La escritura de algoritmos para realizar operaciones sencillas de conteo es una de las primeras cosas que un ordenador puede aprender. Supongamos que se proporciona una secuencia de números, tales como

5 3 0 2 4 4 0 0 2 3 6 0 2

y se desea contar e imprimir el número de ceros de la secuencia.

El algoritmo es muy sencillo, ya que sólo basta leer los números de izquierda a derecha, mientras se cuentan los ceros.

Dicho algoritmo, por consiguiente, utiliza como variable la palabra NÚMERO para los números que se examinan y TOTAL para el número de ceros encontrados. Los pasos a seguir en el algoritmo escrito en un lenguaje natural son:

1. Establecer TOTAL a cero.
2. ¿Quedan más números a examinar?
3. Si no quedan números, imprimir el valor de TOTAL y fin.
4. Si existen más números, ejecutar los pasos 5 a 8.
5. Leer el siguiente número y dar su valor a la variable NÚMERO.
6. Si NÚMERO = 0, incrementar TOTAL en uno.
7. Si NÚMERO \neq 0, no modificar TOTAL.
8. Retornar al paso 2.

El diagrama de flujo se muestra en el DF 2.5

Ejemplo 2.6

Dados tres números, determinar si la suma de cualquier pareja de ellos es igual al tercer número. Si se cumple esta condición, escribir "iguales" y, en caso contrario, escribir "distintas".

Por ejemplo, si los números son:

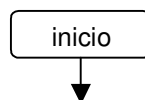
3 9 6

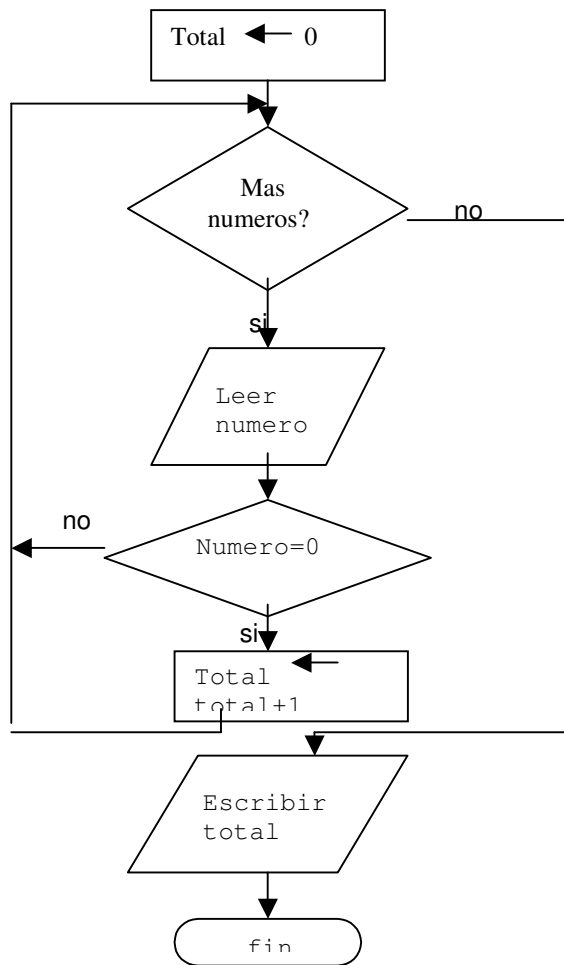
La respuesta es "iguales", ya que $3 \downarrow 6 = 9$. Sin embargo, si los números son:

2 3 4

El resultado será "distintas".

Diagrama de flujo 2.5 (DF 2.5)





Para solucionar este problema, se puede comparar la suma de cada pareja con el tercer número. Con tres números solamente existen tres parejas distintas y el algoritmo de resolución del problema será fácil.

1. Leer los tres valores, A, B y C.
2. Si $A + B = C$ escribir "iguales" y parar.
3. Si $A + C = B$ escribir "iguales" y parar.
4. Si $B + C = A$ escribir "iguales" y parar.
5. Escribir "Distintas" y parar.

El diagrama de flujo correspondiente al algoritmo es el 2.6.

inicio

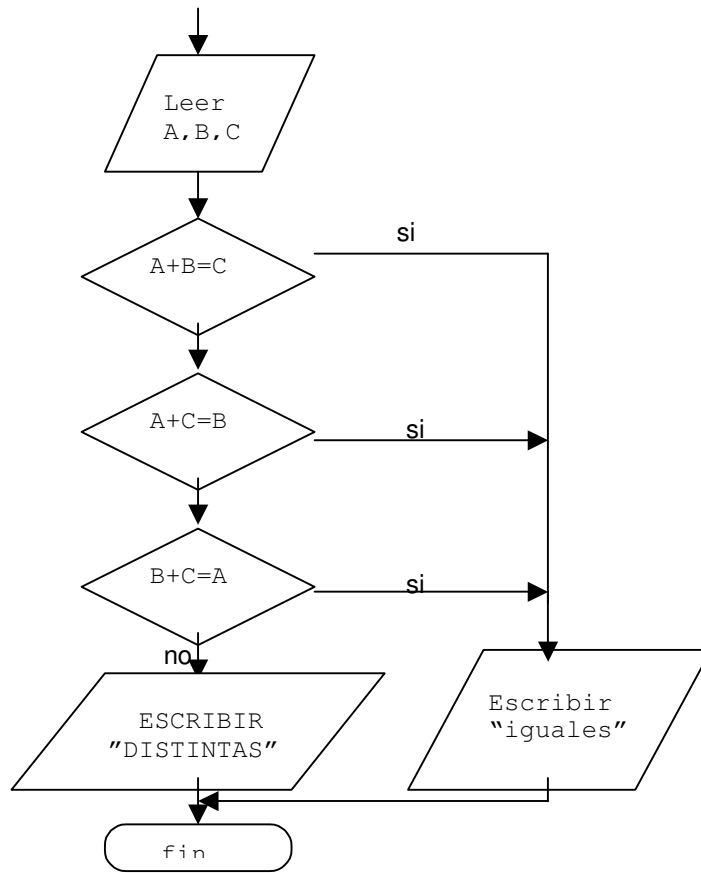


Figura 2.10. Diagrama de flujo 2.6 (Ejemplo 2.6).

2.6 DIAGRAMAS DE NASSI-SCHNEIDERMAN (N-S)

El diagrama N-S de Nassi-Schneiderman – también conocido como 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 escriben en cajas sucesivas y, como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja.

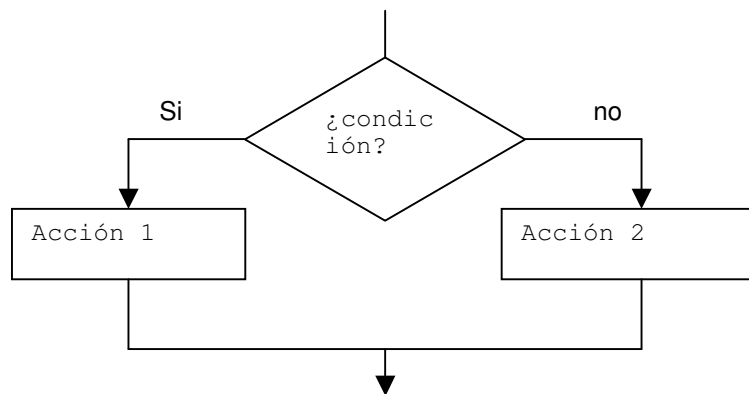
Un algoritmo se representa de la forma siguiente:

Nombre del algoritmo
<Acción 1>
<Acción 2>
<Acción 3>
...
Fin

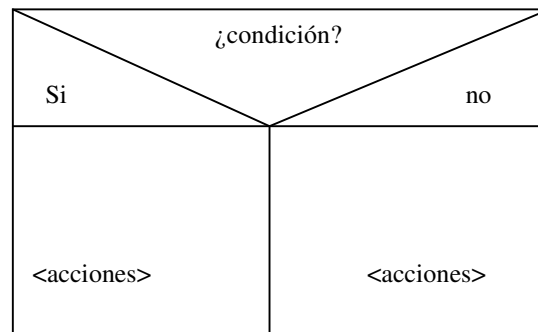
Leer Nombre, horas, precio

Calcular Salario ↓ $\text{horas} * \text{precio}$
Calcular Impuestos ↓ $0.25 * \text{salario}$
Calcular Neto ↓ $\text{salario} - \text{impuesto}$
Escribir Nombre, salario, impuesto, neto

La estructura condicional (el rombo)



Se representa por



Ejemplo 2.7

Se desea calcular el salario neto semanal de un trabajador en función del número de horas trabajadas y la tasa de impuestos:

- Las primeras 35 horas se pagan a tarifa normal.
- Las horas que pasen de 35 se pagan a 1.5 veces la tarifa normal.
- Las tasas de impuestos son:
- Las primeras 60.000 pesetas son libres de impuestos.
- Las siguientes 40.000 pesetas tienen un 25 por 100 de impuestos.
- Las restantes, un 45 por 100 de impuestos.

- La tarifa horaria es 800 pesetas.

Se desea también escribir el nombre, salario bruto, tasas y salario neto (este ejemplo se deja como ejercicio del alumno).

2.7 PSEUDOCÓDIGO

El pseudocódigo es un lenguaje de especificación (descripción) de algoritmos. El uso de tal lenguaje hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil. Los lenguajes APL Pascal y Ada se utilizan a veces como lenguajes de especificación de algoritmos.

El pseudocódigo nació como un lenguaje similar al inglés y era un medio de representar básicamente las estructuras de control de programación estructurada que se verán en capítulos posteriores. Se considera un primer borrador, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación. El pseudocódigo no puede ser ejecutado por una computadora. La ventaja del pseudocódigo es que en su uso, en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico. Es también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa, mientras que en muchas ocasiones suele ser difícil el cambio en la lógica, una vez que esta codificado en un lenguaje de programación. Otra ventaja del pseudocódigo es que puede ser traducido fácilmente a lenguajes estructurados como Pascal, C, Fortran 77/90, Ada 83/95, C++, etc.

El pseudocódigo original utiliza para representar las acciones sucesivas palabras reservadas en inglés –similares a sus homónimas en los lenguajes de programación- tales como start, end, stop, if-then-else, will-end, repeat, until, etc. La escritura de pseudocódigo existe normalmente la indentación (sangría en el margen izquierdo) de diferentes líneas.

La representación en pseudocódigo del diagrama de flujo de la figura 2.7 es al siguiente:

Start

```
//Calculo de ingresos y salarios
read nombre. horas. precio_hora
salario_bruto ↓ horas * precio_hora
tasas ↓ 0.25 * salario_bruto
salario_netto ↓ salario_bruto – tasas
write nombre, salario_bruto, tasas, salario_netto
```

End

El algoritmo comienza con la palabra **start** y finaliza con la palabra **end**, en inglés (en español, **inicio, fin**).entre estas palabras, sólo se escribe una instrucción o acción por línea.

La línea precedida por // se denomina *comentario*. Es una información al lector del programa y no realiza ninguna instrucción ejecutable, sólo tiene efecto de documentación interna del programa. Algunos autores suelen utilizar corchetes y llaves.

No es recomendable el uso de apóstrofes o simples comillas como representan en Basic de Microsoft los comentarios, ya que este carácter es representativo de apertura o cierre de cadenas de caracteres en lenguajes como Pascal y FORTRAN, y daría lugar a confusión.

Otro ejemplo aclaratorio en el uso del pseudocódigo podría ser un sencillo algoritmo del arranque matinal de un coche.

Inicio

```
//arranque matinal de un coche
introducir la llave de contacto
tirar del estrangulador de aire
girar la llave de contacto
pisar el acelerador
oír el ruido del motor
pisar de nuevo el acelerador
```

esperar unos instantes a que se caliente el motor
llevar el estrangulador de aire a su posición

Fin

Por fortuna, aunque el pseudocódigo nació como un sustituto del lenguaje de programación y, pro consiguiente, sus palabras reservadas se conservaron o fueron muy similares a las de dichos lenguajes, prácticamente el inglés, el uso del pseudocódigo se ha extendido en la comunidad hispana con términos en español, como **inicio**, **fin**, **parada**, **leer**, **escribir**, **si-entonces-si_no**, **mientras**, **fin_mientras**, **repetir**, **hasta_que**, etc. Sin duda, el uso de terminología de pseudocódigo en español ha facilitado y facilitará considerablemente el aprendizaje y uso diario de la programación. En esta obra, al igual que en otras nuestras utilizaremos el pseudocódigo en español y daremos en su momento las estructuras equivalentes en inglés, al objeto de facilitar la traducción del pseudocódigo al lenguaje de programación seleccionado.

Así pues, en los pseudocódigos citados anteriormente deberían ser sustituidas las palabras **start**, **end**, **read**, **write por inicio**, **fin**, **leer**, **escribir**, respectivamente.

Inicio	start	leer	read
.			
.			
.			
.			
.			
Fin	end	escribir	write

ACTIVIDADES DE PROGRAMACION RESUELTAS

2.1 Se desea obtener el salario neto de un trabajador conociendo el número de horas trabajadas, el salario, hora y la tasa de impuestos que se le debe reducir.

El algoritmo general es:

1. Obtener valores de horas trabajadas, salario_hora y tasas.
2. Calcular salario_bruto, total de impuestos y salario_netto.
3. Visualizar salario_bruto, total de impuestos y salario_netto.

Las entradas del algoritmo son:

Horas trabajadas, salario_hora, tasas

Las salidas del algoritmo son:

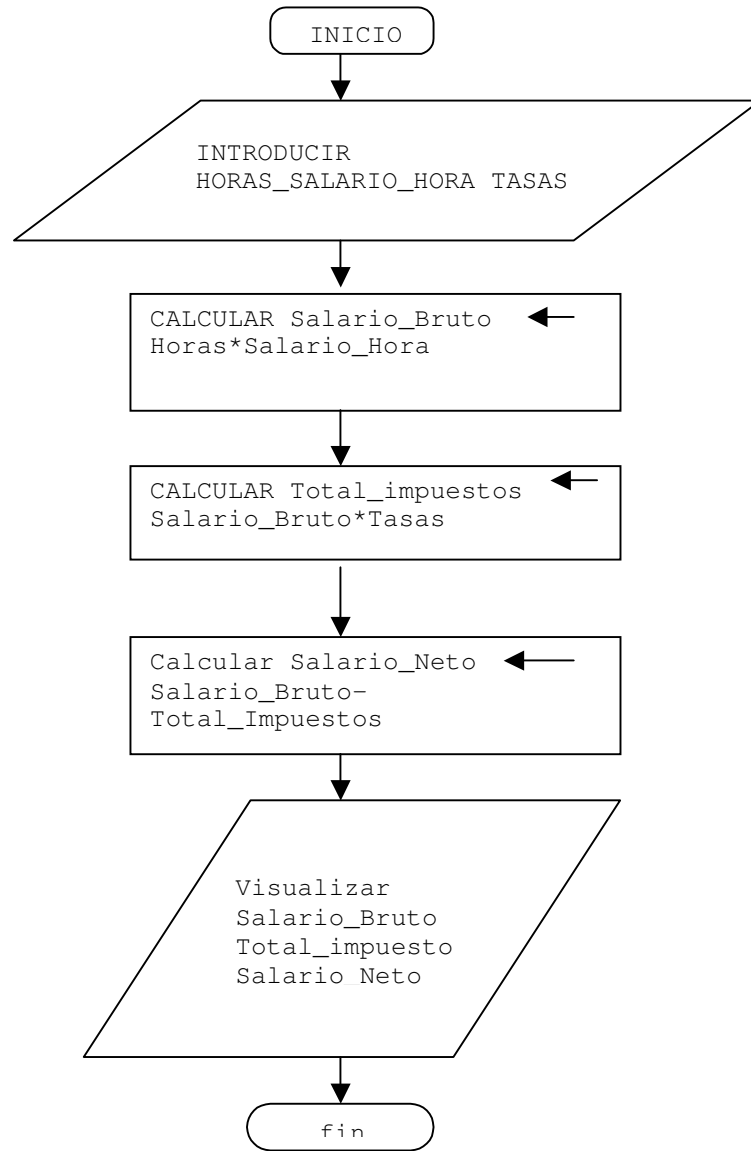
Paga bruta, total de impuestos y paga neta

El refinamiento del algoritmo en pasos de nivel inferior es:

1. Obtener valores de horas trabajada. Salario bruto y tasas.
2. Calcular salario bruto, total de impuestos y paga neta
 - 2.1 Calcular salario bruto multiplicando las horas trabajadas por el salario hora.
 - 2.2 Calcular el total de impuestos multiplicando salario bruto por tasas (tanto por ciento de impuestos).
 - 2.3 Calcular el salario neto restando el total de impuestos de la paga bruta.
3. Visualizar salario bruto, total de impuestos, salario neto.

El diagrama de flujo 2.7 representa este algoritmo

DIAGRAMA DE FLUJO 2.7



2.2 Calcular el valor de la suma $1+2+2+4+\dots+10$

variables CONTADOR (números sucesivos a sumar: 1,2,3,...)
SUMA (totalizador de sumas)

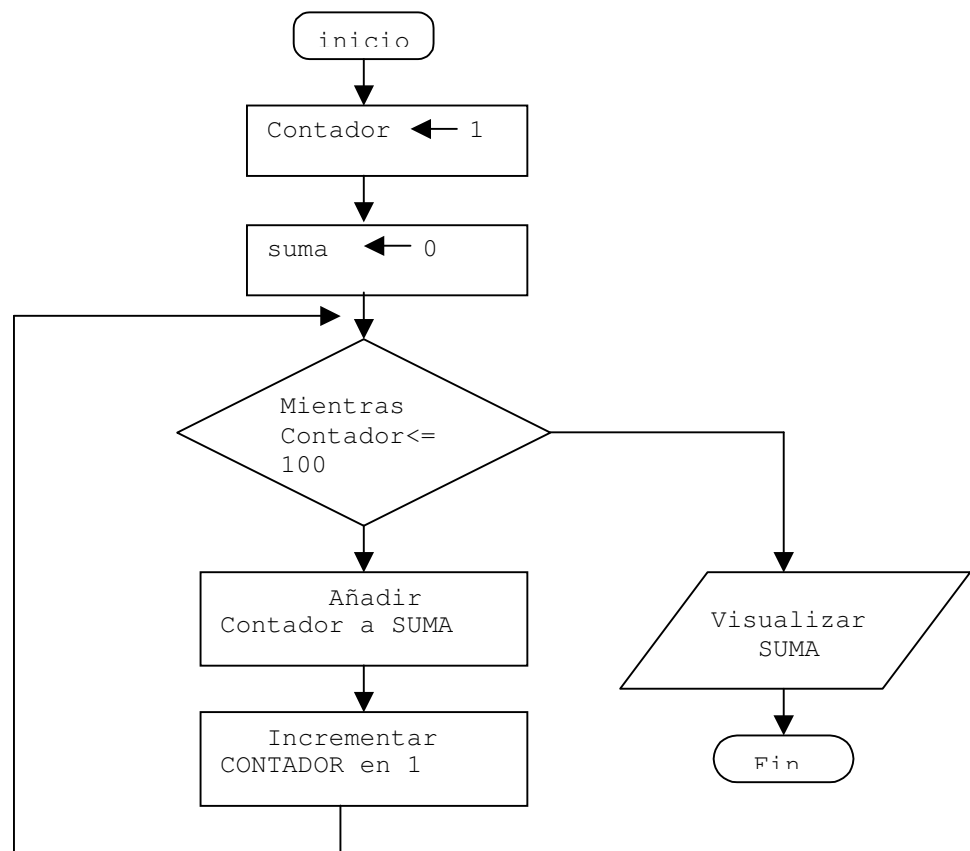
Pseudocódigo

1. inicio.

2. Establecer CONTADOR a 1.
3. Establecer SUMA a 0.
4. **mientras** CONTADOR \leq 100 hacer lo siguiente:
 - 4.1 Sumar CONTADOR a SUMA.
 - 4.2 Incrementar CONTADOR en 1.
- fin_mientras**
5. Visualizar SUMA.
6. Fin.

Este algoritmo se representa por el diagrama de flujo 2.8

Diagrama de flujo 2.8



2.3 Un corredor de maratón (distancia = 42.195 Km.) a recorrido la carrera en 2 horas 25 minutos. Se desea un algoritmo que calcule el tiempo medio en minutos por kilómetro.

El análisis del problema es el siguiente:

Entrada: Cantidad total del tiempo empleado en la carrera; se establece el número total de kilómetros en 42.195, ya que es igual para todos.

Salida: Número medio de minutos por kilómetro, transformando previamente el tiempo total a minutos

Proceso: Dividir el tiempo total en minutos por el número de kilómetros

El pseudocódigo es:

Inicio

Introducir tiempo total
Verificar entrada correcta
Establecer distancia = 42.195 Km.
Pasar tiempo total a minutos
Calcular media tiempo/kilómetro
Escribir resultado

Fin

NOTA: El tiempo T se ha de convertir en minutos, bien tras introducir su valor o en la instrucción 4 (pasar tiempo total a minutos). La fórmula de conversión es:

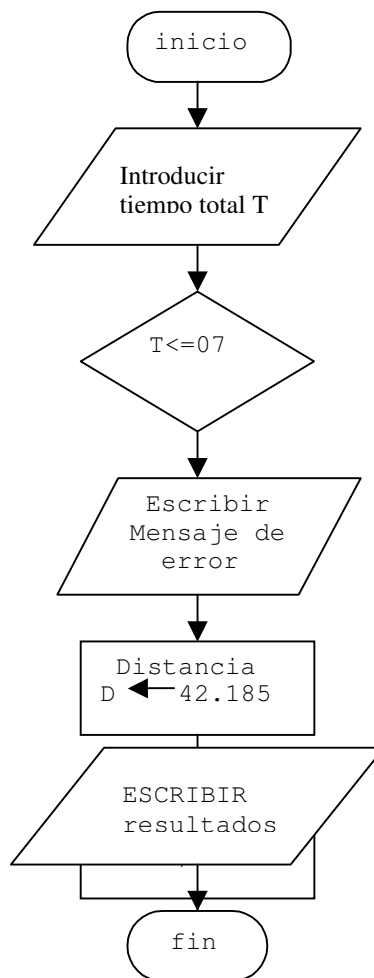
$$T = 2 * 50 + 25$$

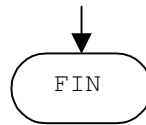
O con carácter general, si el tiempo es h horas m minutos

$$T = h * 60 + m$$

El diagrama de flujo de este algoritmo es el representado a continuación.

Diagrama de flujo 2.9





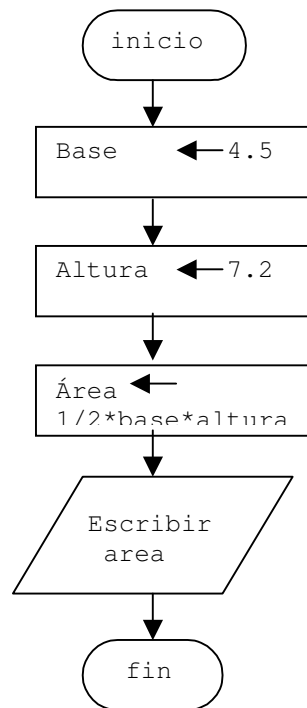
2.5 Escribir un algoritmo para calcular el área de un triángulo dada la base y la altura.

Análisis

La formula geométrica del área o superficie del triángulo es:

$$S = (1/2) B \cdot H \quad B = \text{base} \quad H = \text{altura}$$

Se necesita asignar los valores de la base y la altura a variables –por ejemplo, BASE y ALTURA. Respectivamente -; a continuación, calcular el área utilizando la formula y asignar los resultados del calculo a una variable llamada AREA. Supongamos B=4.5 y H=7.2



Ejercicios

2.1 deducir los resultados que se obtienen del siguiente algoritmo:

```

ver entero: x,y,z
inicio
  x ↓ 5
  y ↓ 20
  z ↓ x-y
escribir (x,y)
  
```

escribir (z)
fin

2.2 ¿Qué resultados producirá este algoritmo?

Var. entero; nx, doble

inicio

Nx ↓ 25
Doble ↓ NX * 2
Escribir (NX)
Escribir (doble)

fin

2.3 Escribir un algoritmo que calcule y escriba el cuadrado de 243

2.4 Escribir un algoritmo que lea un número y escriba su cuadrado.

2.5 Determinar el área y volumen de un cilindro cuyas dimensiones radio y altura se leen desde el teclado.

2.6 Calcular el perímetro y la superficie de un cuadrado dada la longitud de su lado.

2.7 Realizar el algoritmo que suma dos números.

2.8 Calcular la superficie de un círculo.

2.9 Calcular el perímetro y la superficie de un rectángulo dadas la base y la altura del mismo.

2.10 Escribir un algoritmo que lea un número de una marca de automóviles seguida del nombre de su modelo e informe del modelo seguido del nombre.

2.11 Determinar la hipotenusa de un triángulo rectángulo conocida las longitudes de los catetos.

2.12 Diseñar un algoritmo que realice las siguientes conversiones: una temperatura dada en grados Celsius a grados Fahrenheit.

NOTA: La forma de conversión es $F = (9/5) C + 32$

2.13 Diseñar un algoritmo que calcule el área de un triángulo en función de las longitudes de sus lados:

$$\text{AREA} = \sqrt{p(p-a)(p-b)(p-c)}$$

donde $p = (a + b + c) / 2$ (semiperímetro).

2.14 Se desea un algoritmo para convertir metros a pies y pulgadas (1 metro = 39.37 pulgadas, 1 pie = 12 pulgadas).

2.15 El cambio de divisas en la Bolsa de Madrid el día 25 de Agosto de 1987 fue el siguiente:

100 chelines austriacos	= 956,871 pesetas.
1 dólar EE.UU.	= 122,499 pesetas
100 dracmas griegas	= 88,607 pesetas.
100 francos belgas	= 323,728 pesetas.
1 franco francés	= 20,110 pesetas
1 libra esterlina	= 178,938 pesetas.
100 liras italianas	= 9,289 pesetas.

2.16 Desarrollar algoritmos que realicen las siguientes conversiones:

- Leer una cantidad en chelines austriacos e imprimir el equivalente en pesetas.
- Leer una cantidad en dracmas griegos e imprimir el equivalente en francos franceses.
- Leer una cantidad en pesetas e imprimir el equivalente en dólares y en liras italianas.

