CAPITULO : 2

LA RESOLUCION DE PROBLEMAS CON COMPUTADORAS Y LAS HERRAMIENTAS DE PROGRAMACIÓN

Contenido:

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.*

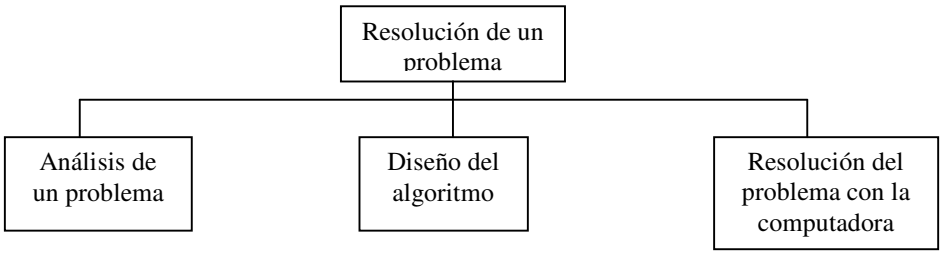


Figura 2.1 la resolución de un problema

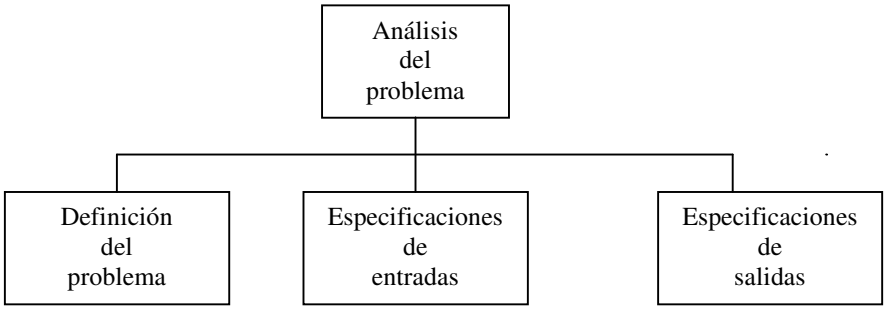


Figura 2.2 Análisis del problema

Ejemplo 2.1.

Leer el radio de un circulo 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 circulo. 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 circulo(variable RADIO).
Salidas:	superficie del circulo(variable Area). Circunferencia del circulo(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 circulo 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.

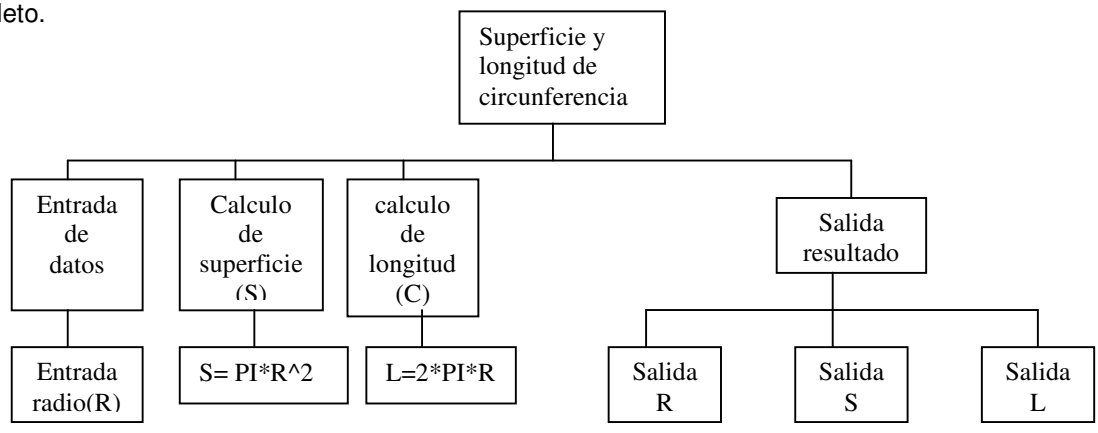


Figura 2.3. refinamiento de un algoritmo

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
leer radio	leer radio
calcular superficie	superficie=3.141592 * radio^2
calcular circunferencia	circunferencia 2 * 3.141592 * radio
escribir resultados	*escribir* 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.

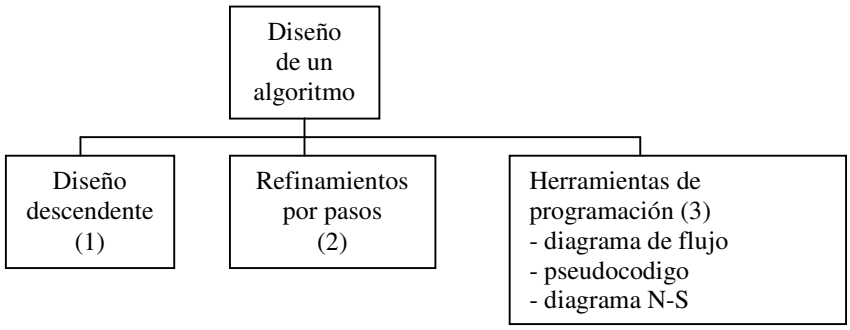


FIGURA 2.4 FASES DEL DISEÑO DE UN ALGORITMO

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.

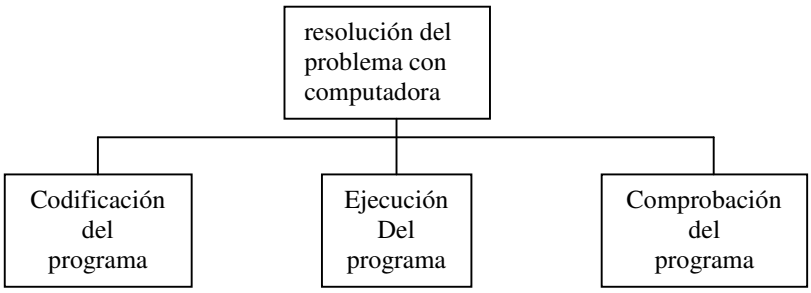


Figura 2.5 resolución del problema mediante computadora

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 .

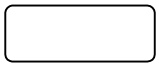
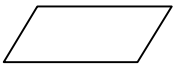
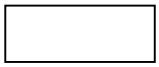
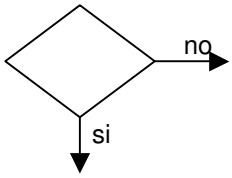
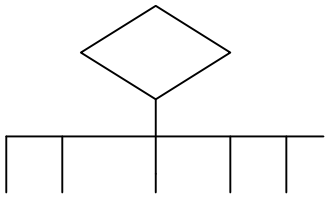
2.5.1. DIAGRAMAS DE FLUJO

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 trabajor 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).

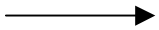
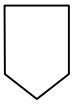
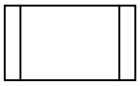
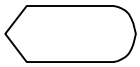
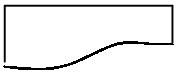
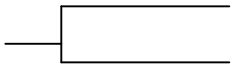
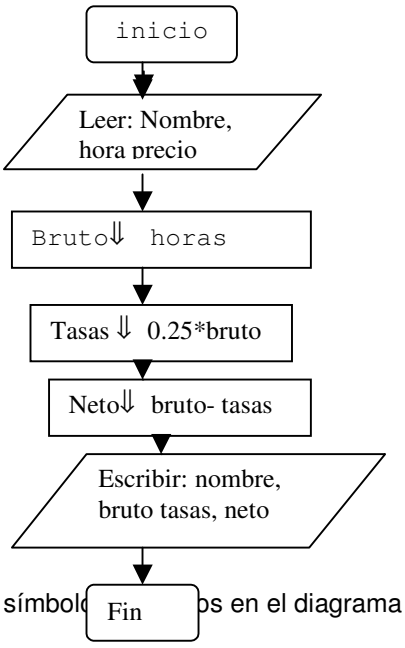
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones).
	Línea conectora (sirve de unión entre dos símbolos).
	Conector (conexión entre dos puntos del organigrama situado en paginas diferentes).
	Llamada a subrutina o a un proceso predeterminado (una subrutina es 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 simbo_

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

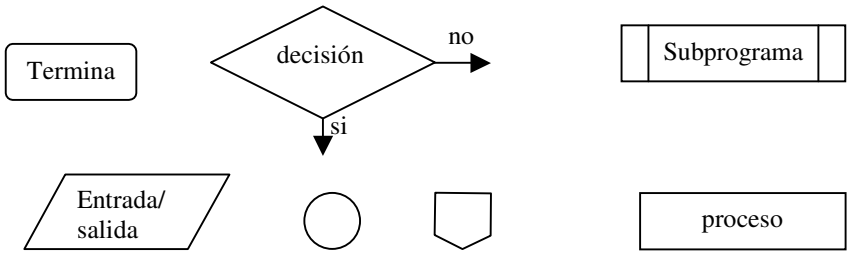


Figura 2.9 símbolos utilizados en los diagramas de flujos

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* (◊) representa el flujo secuencial de la lógica del problema.

Un rectángulo (▭) 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 (▱) 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 (◇) 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 (▭).

Un pequeño círculo (◯) 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 trapezoide (⏢) 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