

Tema 1: Algoritmos y programas

Objetivo: utilizar la computadora como una herramienta para resolver problemas.

Medio: Aprendizaje de lenguajes y técnicas de programación.

La resolución de un problema exige al menos los siguientes pasos:

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

Un algoritmo es un método para resolver un problema. Esta palabra proviene del matemático persa Mohammed al-Khowâritzmî, que vivió durante el siglo IX. Euclides, inventor de un método para calcular el M.C.D. de dos números, fue el otro padre de la algoritmia (siglo IX A.C.).

El profesor Niklaus Wirth tituló uno de sus más famosos libros “Algoritmos + Estructuras de datos = Programas”, enunciado que aplicaremos de aquí en adelante.

Los sistemas de procesamiento de la información

Un ordenador es un sistema de procesamiento de información. Es un sistema porque es un conjunto de componentes conectados e interactivos que tienen un propósito y una unidad total, procesa porque es capaz de transformar unos datos de entrada en los resultados buscados, e información porque maneja conjuntos ordenados de datos.

Un sistema de proceso de la información tiene tres componentes muy claros:

Entrada = datos → PROCESO → Salida = información

Para procesar la información está el hardware y el software:

1. Hardware: conjunto de componentes físicos de una computadora: CPU, Memoria principal, dispositivos de almacenamiento y dispositivos de E/S.
2. Software: Conjunto de programas que sirven para manejar el hardware.

Concepto de algoritmo

El objetivo principal de la asignatura es resolver problemas mediante una computadora. Por lo tanto, un programador es principalmente una persona que resuelve problemas, y debemos aprender a hacerlo de un modo riguroso y sistemático. Denominamos *metodología de la programación* al conjunto de métodos necesarios para resolver problemas mediante programas. La resolución de un problema requiere el diseño de un algoritmo:

Problema → Diseño del algoritmo → Programa de computadora

Los pasos para la resolución del problema son:

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

Entonces, un algoritmo es una fórmula para resolver un problema. Es un conjunto de acciones o secuencia de operaciones que ejecutadas en un determinado orden resuelven el problema. Para resolver un problema pueden existir N algoritmos, hay que encontrar el más efectivo.

Los algoritmos son independientes del lenguaje de programación utilizado y de la computadora donde se ejecutan.

Características de los algoritmos

Las características que debe cumplir todo algoritmo son:

- Debe ser preciso e indicar el orden de realización de cada paso.
- Debe estar bien definido. Si se sigue un algoritmo dos veces se debe obtener el mismo resultado cada vez
- Debe ser finito. Si se sigue un algoritmo se debe terminar en algún momento, o sea, debe tener un número finito de datos.

La definición de un algoritmo debe describir tres partes: Entrada, Proceso y Salida. Tomemos como ejemplo una receta de cocina:

- Entrada: ingredientes y utensilios empleados
- Procesos: elaboración de la receta en la cocina
- Salida: terminación del plato

Los lenguajes de programación

Para resolver el problema, el procesador debe ser capaz de interpretar el algoritmo (comprender las instrucciones de cada paso y realizar las operaciones correspondientes). Cuando el procesador es una computadora el algoritmo se ha de expresar en un formato que se denomina programa. Un programa se describe en un *lenguaje de programación* y las operaciones que transforman un algoritmo en un programa se llaman *programación*. Por lo tanto un *programador* es una persona que escribe y diseña programas.

Una posible clasificación de los lenguajes de programación puede ser por la proximidad al usuario o a la máquina:

- Lenguaje máquina
- Lenguajes de bajo nivel (ensamblador)
- Lenguajes de alto nivel

Instrucciones a la computadora

Los diferentes pasos de un algoritmo se expresan en los programas como instrucciones, sentencias o proposiciones.

Las instrucciones básicas y comunes a (casi) todos los lenguajes de programación se pueden clasificar de la siguiente manera:

- Instrucciones de E/S.
- Instrucciones aritmético-lógicas.
- Instrucciones selectivas.
- Instrucciones repetitivas.

Lenguajes máquina

Los lenguajes máquina son aquellos que están escritos en lenguajes directamente inteligibles por la máquina. Sus instrucciones son cadenas binarias (ceros y unos).

Las instrucciones en lenguaje máquina dependen del hardware de la computadora.

Ventajas:

- Cargar un programa en memoria sin necesidad de traducción.
- En parte como consecuencia de la anterior, aumento de velocidad.

Inconvenientes:

- Dificultad y lentitud en la codificación.
- Poca fiabilidad.
- Dificultad para verificar y poner a punto los programas.
- Los programas sólo son ejecutables en el mismo procesador

Lenguajes de bajo nivel

Los lenguajes de bajo nivel (el ensamblador) son más fáciles de utilizar que los L. M., ya que utilizan mnemotécnicos para evitar las cadenas binarias.

Un programa en lenguaje ensamblador no es directamente ejecutable por la computadora, si no que requiere una fase de traducción a lenguaje máquina. El código original se denomina *programa (código) fuente* y el código traducido se llama *programa (código) objeto*. El traductor de programas fuente a objeto se llama *ensamblador*.

Ventajas:

- Velocidad.
- Más fáciles de codificar que en lenguaje máquina.

Inconvenientes:

- Dependencia total de la máquina.
- La formación es más complicada que la correspondiente a los programadores de alto nivel, ya que además de las técnicas de programación imprescindibles se ha de conocer el interior de la máquina.

Lenguajes de alto nivel

Los lenguajes de alto nivel están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil. Son, por lo tanto, mucho más cercanos a la persona y mucho más lejanos a la máquina.

Ventajas:

- Aprendizaje más sencillo.
- La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos.
- El mantenimiento de los programas es mucho más sencillo.
- Reducción en el coste de los programas.
- Portabilidad.

Inconvenientes:

- Los programas necesitan diferentes traducciones antes de poder ser ejecutados.
- No se aprovechan al 100% los recursos internos de la máquina.
- Aumento de la ocupación de memoria.
- Lentitud.

Algunos ejemplos de lenguajes de alto nivel son: C, C++, COBOL, FORTRAN, BASIC, Pascal, Visual Basic, Clipper, Ada, Modula-2, Prolog, LISP, Smalltalk, Delphi, Miranda, Eiffel, ...

Traductores de lenguaje

Los traductores de lenguaje son programas que traducen los programas fuente escritos en lenguajes de alto nivel a código máquina.

Se dividen en:

- Interpretes: Toma una instrucción, la traduce y a continuación la ejecuta, para continuar posteriormente con la siguiente. Ejemplos: BASIC, Smalltalk, ...
- Compiladores: Traduce completamente el código fuente a lenguaje máquina, para que puedan ser ejecutados posteriormente. Ejemplos: C, Pascal, ...

Los programas compilados pueden conseguir una mayor eficiencia, en cambio los interpretados suelen ser más flexibles.

Las fases seguidas para compilar un programa son las siguientes:

Programa fuente → Compilador → Programa objeto → Montador → Programa en LM

Clasificación de los lenguajes de programación

Podemos clasificar también los lenguajes de programación según el paradigma utilizado

- Imperativos
- Declarativos
 - Funcionales
 - Lógicos
- Lenguajes orientados a objetos
- Lenguajes de 4ª generación

Datos, tipos de datos y operaciones primitivas

Dato: expresión general que describe los objetos con los cuales opera una computadora.

Los algoritmos y programas operan sobre datos.

Podemos clasificar los tipos de datos:

- Simples (sin estructura)
 - Numéricos: enteros, reales
 - Lógicos: booleanos
 - Caracteres: carácter, cadena
- Compuestos (estructurados)

También admiten otra clasificación:

- Estándar
- No estándar

Datos numéricos

Entero: Subconjunto finito de los números enteros (Z).

Real: Subconjunto de los números reales (R). El mayor problema que acarrea el manejo de datos numéricos reales es la precisión. Números muy grandes o muy pequeños sufren un redondeo que puede hacer variar su valor significativamente.

Un formato habitual de representación de números reales es la notación exponencial o científica: $x.yyyyyy * 10^z$.

Datos lógicos

El tipo lógico o booleano es aquel que sólo puede tomar uno de dos valores: cierto (true o verdadero) y falso (false).

Es, por ejemplo, el valor resultante en cualquier comparación.

Datos tipo carácter y cadena

Carácter: Uno solo del conjunto finito y ordenado de caracteres que reconoce la computadora (letras, dígitos, caracteres especiales, ASCII) entre comillas simples.

Tipo de cadena o String: Conjunto de caracteres entre comillas dobles.

Constantes y variables

Constante: Valor que no cambia durante la ejecución del programa.

Variable: Objeto o partida de datos cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa.

Tanto constantes como variables tienen una serie de características comunes:

- Un nombre que los diferencia del resto, llamado identificador.
- Un tipo que nos determina las operaciones que podemos hacer con ese dato.
- Un valor que puede variar o no a lo largo de la operación.

Una variable de un cierto tipo solamente puede tomar valores de su mismo tipo. Si se intenta asignar un valor que no corresponde a su tipo se producirá un error de tipo.

Los identificadores válidos son aquellos que empiezan con un carácter alfabético seguido de 0 o más caracteres alfanuméricos o ‘underscores’ (guiones bajos o símbolos de subrayado). No se diferencian mayúsculas y minúsculas. Además, no se podrán tomar como identificadores aquellas palabras que coincidan con instrucciones de control o funciones internas (palabras reservadas).

Una constante no identificada (que no tiene nombre) es un *literal*.

La ventaja de usar constantes con nombre es que en cualquier lugar donde quiera que vaya la constante, basta con poner su nombre y luego el compilador lo sustituirá por su valor.

Relación entre variables y constantes en memoria:

- Al detectar una variable o una constante con nombre, automáticamente se reserva en memoria espacio para guardar esa variable o constante. El espacio reservado depende del tipo de la variable.
- En esa zona de memoria es en la que se guarda el valor asociado a la variable o constante y cuando el programa use esa variable, ira a esa zona de memoria a buscar su valor.

Expresiones y operadores

Una expresión es una combinación de constantes, variables, signos de operación, paréntesis y nombres especiales (nombres de funciones estándar), con un sentido unívoco y definido y de cuya evaluación resulta un único valor.

Una expresión consta de operandos y operadores. Toda expresión tiene asociada un tipo que se corresponde con el tipo del valor que devuelve la expresión cuando se evalúa, por lo que habrá tantos tipos de expresiones como tipos de datos.

Expresiones aritméticas

Los operandos deben ser numéricos y las operaciones aceptadas son las siguientes, ordenados por orden de precedencia:

- Menos unario (-, símbolo para representar números negativos)
- Potencia (^ ó **).
- Multiplicación (*), división real (/), división entera (div), módulo (mod).
- Suma (+), Resta (-).

Entre dos operaciones que tienen la misma precedencia para resolver la ambigüedad, hay que usar la regla de la asociatividad. La más normal es la de la asociatividad a izquierdas (primero lo de la izquierda).

Al igual que en matemáticas, las expresiones que están encerradas entre paréntesis se evalúan primero. Si existen varios niveles de paréntesis, las expresiones más internas se evalúan primero.

Expresiones lógicas

Una expresión lógica es aquella que sólo puede devolver dos valores (Verdadero o Falso). Los valores que pueden aparecer en una expresión lógica son de 2 tipos: lógicos y relacionales.

La particularidad de las expresiones lógicas es que mientras en una expresión numérica por devolver un valor numérico los operandos solo pueden ser números, en una expresión lógica los operandos no tienen porque ser booleanos aunque se devuelva un valor booleano. Esto es lo que ocurre cuando en la expresión lógica utilizamos

operadores relacionales con lo cual se obtienen valores lógicos o booleanos a partir de otros que no lo son. En cambio cuando los operadores son lógicos los operandos obligatoriamente también tienen que ser lógicos.

Operadores relacionales: >, <, >=, <=, =, <> (también !=).

¿Cómo se evalúa una expresión relacional?:

- Primero se evalúa el primer operando y se sustituye por su valor.
- Luego se evalúa el segundo operando y se sustituye por su valor.
- Finalmente se aplica el operador relacional y se devuelve el valor booleano correspondiente.

Se pueden realizar comparaciones entre caracteres, para ello hay que considerar que el orden establecido es el seguido en el código ASCII.

Operadores lógicos: Y (and, &), O (or, |), NO (not, ~).

Y, O, son operadores binarios (necesitan 2 operandos de tipo lógico). El No es unario y se coloca primero el No, y después el operando.

Tablas de verdad de los operadores lógicos Y y O:

Operando 1	Operando 2	Op1 Y Op2	Op1 O Op2
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

Tabla de verdad del operador NO:

Operando	NO operando
V	F
F	V

Prioridades de los operadores:

- Lo más prioritario es el NO
- Luego el Y y el O.
- Operadores relacionales

Funciones internas

Son funciones matemáticas diferentes de las operaciones básicas pero que se incorporan al lenguaje y que se consideran estándar.

Abs (x)

Arctan (x)

Cos (x)

Sen (x)

Exp (x)

Ln (x)

Log 10 (x)

Redondeo (x)

Trunc (x)

Cuadrado (x)

Raíz (x)

La operación de asignación

Consiste en atribuir un valor a una variable. El valor será una expresión (constante, variable,...). El símbolo utilizado es ':=' , aunque también se suele utilizar '←'.

Variable := expresión

El proceso de asignación se realiza en 2 fases:

- Se evalúa la expresión de la parte derecha de la asignación obteniéndose un único valor.
 - Se asigna ese valor a la variable de la parte izquierda.
- ¿Qué es lo que hay que tener en cuenta?:
- En la parte izquierda sólo puede haber una variable.
 - La variable a la que se le asigna el valor pierde su valor anterior.
 - La variable que aparece en la derecha ya que como se evalúa primero la de la derecha cuando se tenga que evaluar el valor de esa variable se tomara su valor antiguo.
 - EL TIPO DEL VALOR QUE SE OBTIENE AL EVALUAR LA PARTE DERECHA TIENE QUE SER EL MISMO QUE EL TIPO DE LA VARIABLE DE LA PARTE IZQUIERDA, ES DECIR A UNA VARIABLE SOLO SE LE PUEDEN DAR VALORES DE SU MISMO TIPO.

Conversión de tipo

En las asignaciones no se pueden asignar valores a una variable de un tipo diferente del suyo. Se puede dar una excepción al asignar a una variable real un valor entero, ya que los enteros forman parte del conjunto de valores reales (aunque algunos lenguajes no realizan esto automáticamente).

Existen dos funciones internas que nos permiten realizar conversiones de tipo:

Chr (X): Devuelve el carácter correspondiente en el código ASCII al valor entero X.

Ord (C): Devuelve el código ASCII correspondiente al carácter C.

Entrada y salida de información

Las dos operaciones básicas de cada salida son las de lectura y de escritura.

La *lectura* es equivalente a la asignación en cuanto que va a haber una variable que recibe un valor, pero este valor no resulta de evaluar ninguna expresión, sino que lo vamos a leer de un dispositivo externo de entrada (puede que no sea el teclado).

Leer (nombre de la variable)

El valor introducido por el dispositivo externo, tiene que ser del mismo tipo del que la variable que se le asigne, siguiendo las normas de conversión de tipo. En caso contrario se producirá un error.

La operación de *escritura* muestra el valor de una variable, constante o literal en un dispositivo externo de salida, en nuestro caso (casi siempre) la pantalla.

Escribir (lista de expresiones)

Escribir (expr1, expr2, ..., exprN)

Tema 2: La resolución de problemas con las computadoras y las herramientas de la programación

Resolución de problemas

La resolución de un problema desde el punto de vista algorítmico tiene 3 fases:

- Análisis del problema: Comprensión.
- Diseño del algoritmo: Resolución algorítmica.
- Resolución en computadora: Implantación del algoritmo en un lenguaje de programación.

Análisis del problema

El objetivo de ésta fase es comprender el problema. El problema debe estar bien definido si queremos llegar a una solución satisfactoria.

Para poder definir correctamente el problema necesitamos detallar:

- Definición del problema: ¿Cuál es el problema?
- Especificaciones de entrada: ¿Qué datos se necesitan para resolver el problema?
- Especificaciones de salida: ¿Qué información debe proporcionar la resolución del problema?

Diseño del algoritmo

Una vez comprendido el problema debemos determinar que pasos o acciones tenemos que realizar para resolverlo. Una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutar por la máquina constituyen el algoritmo.

Los problemas complejos se pueden resolver más eficazmente con la computadora cuando se rompen en subproblemas que sean más fáciles de solucionar que el original.

Esta descomposición sucesiva del problema original en subproblemas más simples hasta que pueden ser implementados se denomina **diseño descendente** (top-down design).

- Los pasos diseñados en el primer esbozo del algoritmo son incompletos e indicarán sólo unos pocos pasos.
- Posteriormente éstos se amplían en una descripción más detallada con más pasos específicos. Este proceso se denomina refinamiento del algoritmo (stepwise refinement).
- Se detiene el refinamiento cuando la solución a cada uno de los pequeños problemas es suficientemente simple.

Las ventajas de aplicar el diseño descendente son:

- Al dividir el problema en módulos o partes se comprende más fácilmente.
- Al hacer modificaciones es más fácil sobre un módulo en particular que en todo el algoritmo.
- En cuanto a los resultados, se probarán mucho mejor comprobando si cada módulo da el resultado correcto que si intentamos probar de un golpe todo el programa porque si se produce un error sabemos en que módulo ha sido.

Tras estos pasos es necesario representar el algoritmo mediante alguna herramienta de programación.

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.

Al realizar esto debemos seguir una serie de reglas:

- Las acciones o pasos a realizar tienen que tener un determinado orden.
- En cada momento solo se puede ejecutar una acción.

El flujo de control usual de un algoritmo es secuencial, es decir, se siguen las instrucciones de arriba a abajo.

Ejemplo: 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:

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.

Como hemos dicho anteriormente, el algoritmo general se descompondrá en pasos más simples mediante refinamiento sucesivo. Cada acción puede descomponerse a su vez en otras acciones simples. Así, por ejemplo, un primer refinamiento del algoritmo (módulo o sub-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. finalizar
 - si no
 - 3.3. ir al cine
 - fin si
4. si hay cola entonces
 - 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

Existen diferentes aspectos a considerar:

1.- En primer lugar, ciertas palabras reservadas se han remarcado deliberadamente (mientras, si no, etc..). Estas palabras describen las estructuras de control fundamentales que se encuentran en casi todos los algoritmos.

Son principalmente las siguientes:

- selección (expresadas por, si-entonces-si-no, if-then-else, ...)
- repetición (expresadas con mientras-hacer, repetir-hasta, while-do, ...)

2.- Otro aspecto a considerar es el empleo de indentación (sangrado o justificación) en la escritura de algoritmos. Es importante la escritura de programa como su posterior lectura. Esto se facilita con la indentación de las acciones interiores a las estructuras fundamentales citadas: selectivas y repetitivas.

Para terminar ejemplo, describiremos las acciones necesarias para refinar una parte del algoritmo. Analicemos la acción 5.3. *Localizar las(s) butaca(s)*

1. inicio //algoritmo para encontrar la butaca del espectador
2. caminar hasta llegar a la primera fila de butacas
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 numero de butaca no coincida con numero de billete hacer
avanzar a través de la fila a la siguiente butaca
fin mientras
5. sentarse en la butaca
6. fin

Resolución del problema mediante computadora

Una vez que el algoritmo está diseñado y representado gráficamente mediante una herramienta de programación se debe pasar a la fase de resolución práctica del problema con la computadora.

Esta fase se descompone a su vez en las siguientes subfases:

- *codificación* del algoritmo en un programa
- *ejecución* del programa
- *comprobación* del programa

Representación gráfica 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.

Los métodos usuales para representar un algoritmo son:

- diagrama de flujo
- diagrama N-S (Nassi-Schneiderman)
- lenguaje de especificación de algoritmos: pseudocódigo
- lenguaje natural
- fórmulas



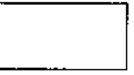
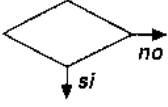
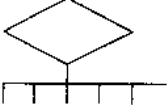

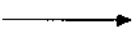


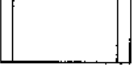
Los métodos 4 y 5 no suelen ser fáciles de transformar en programas. Una descripción en español narrativo no es satisfactoria, ya que puede ser ambigua y poco definida. Una fórmula es buen sistema de representación, sin embargo, no es frecuente que un algoritmo pueda ser expresado por medio de una simple fórmula.


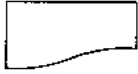
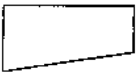
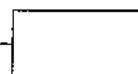
Diagramas de flujo

Un diagrama de flujo (flowchart) es una de las técnicas de representación de algoritmos más antigua y a la vez más utilizada. Su empleo ha disminuido considerablemente desde la aparición de lenguajes de programación estructurados, ya que no es una técnica estructurada.

Un diagrama de flujo es un diagrama que utiliza unos símbolos estándar (cajas) 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.

Los símbolos estándar normalizados por ANSI (abreviatura de American National Standards Institute) son:

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 cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas —respuestas SI o NO—, pero puede tener tres o más, 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 organograma a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página 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 organograma situado en páginas diferentes).
	Llamada a subrutina o a un proceso predeterminado (una subrutina es un módulo 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 E/S).
	Teclado (se utiliza en ocasiones en lugar del símbolo de E/S).
	Comentarios (se utiliza para añadir comentarios clasificadores a otros símbolos del diagrama de flujo. Se pueden dibujar a cualquier lado del símbolo).

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 representan el flujo secuencial de la lógica del programa.
- 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 trapecio 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.

Diagramas de Nassi-Schneiderman (N-S)