Programación

Fundamentos de programación

Unidad 1

Jesús Alberto Martínez
versión 0.2

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante	Atención	Interesante

Unidad 1. Fundamentos de programación

1 INTRODUCCIÓN

La razón principal por la que una persona utiliza un ordenador es para resolver problemas (en el sentido más general de la palabra), o en otras palabras, procesar una información para obtener un resultado a partir de unos datos de entrada.

Los ordenadores resuelven los problemas mediante la utilización de programas escritos por los programadores. Los programas de ordenador no son entonces más que métodos para resolver problemas. Por ello, para escribir un programa, lo primero es que el programador sepa resolver el problema que estamos tratando.

El programador debe identificar cuáles son los datos de entrada y a partir de ellos obtener los datos de salida, es decir, la solución, a la que se llegará por medio del procesamiento de la información que se realizará mediante la utilización de un método para resolver el problema que denominaremos algoritmo.

2 ALGORITMO

Por algoritmo entendemos un conjunto ordenado y finito de operaciones que permiten resolver un problema que además cumplen las siguientes características:

- Tiene un número finito de pasos
- Acaba en un tiempo finito. Si no acabase nunca, no se resolvería el problema.
- Todas las operaciones deben estar definidas de forma precisa y sin ambigüedad.
- Puede tener varios datos de entrada y como mínimo un dato de salida.

Un claro ejemplo de algoritmo es una receta de cocina, donde tenemos unos pasos que hay que seguir en un orden y deben de estar bien definidos, tiene un tiempo finito y tiene unos datos de entrada (ingredientes) y una salida (el plato).

Por ejemplo, el algoritmo para freír un huevo podría ser el siguiente:

Datos de entrada: Huevo, aceite, sartén, fuego.

Datos de salida: huevo frito.

Procedimiento:

1. Poner el aceite en la sartén.
2. Poner la sartén al fuego.
3. Cuando el aceite esté caliente, cascar el huevo e introducirlo.

4. Cubrir el huevo de aceite.
5. Cuando el huevo esté hecho, retirarlo.

La codificación de un algoritmo en un ordenador se denomina ***programa***.

3 Lenguajes de programación

El algoritmo anterior lo hemos expresado en lenguaje natural, en nuestro idioma, pero cuando realizamos un algoritmo destinado a realizar un proceso en un ordenador ya hemos visto que tenemos que realizar un programa.

El lenguaje natural es impreciso, ambiguo, no podemos realizar el programa en este lenguaje, tenemos que **limitar** lo que podemos decirle al ordenador, y cómo decirselo. Esta limitación son los lenguajes de programación, que de una forma más precisa, sin ambiguedades, permiten poder indicar ese algoritmo de una forma finita y con un final previsible.

3.1 Tipos de lenguajes de programación

Los lenguajes de programación se pueden dividir de varias formas.

Según su proximidad a la máquina:

- Lenguajes de bajo nivel. Lenguajes diseñados para un hardware específico, no se pueden migrar a otro hardware. Sacan un mayor rendimiento de ese hardware. La forma de expresar y representar las instrucciones son muy cercano al hardware. Se consideran de bajo nivel los lenguajes de programación máquina y ensamblador.

```
; Hola Mundo en ensamblador
 .model small
 .stack
 .data
    saludo   db "Hola mundo!!!", "$"

 .code

 main  proc              ;Inicia proceso
    mov   ax,seg saludo     ;hmm ¿seg?
    mov   ds,ax          ;ds = ax = saludo
```

```
        mov   ah,09           ;Function(print string)
        lea   dx,saludo          ;DX = String terminated by "$"
        int   21h                ;Interruptions DOS Functions

    ;mensaje en pantalla

        mov   ax,4c00h        ;Function (Quit with exit code (EXIT))
        int   21h             ;Interruption DOS Functions

    main  endp                ;Termina proceso
    end main
```

- Lenguajes de alto nivel. Lenguajes que son independientes del hardware donde se ejecutan. La forma de expresar las instrucciones es más parecida al lenguaje natural, usando una mezcla de ingles y matemáticas. C, C#, Java, Python, PHP, etc.....

```
/* Programa C: Hola mundo */

#include <conio.h>
#include <stdio.h>

int main()
{
    printf( "Hola mundo." );

    getch(); /* Pausa */

    return 0;
}
```

4 CICLO DE VIDA DE UN PROGRAMA

La creación de cualquier programa (software o sw) implica la realización de tres pasos genéricos:

- Definición ¿Qué hay que desarrollar?
- Desarrollo.
- Mantenimiento.

4.1 Fase de definición

Se intenta caracterizar el sistema que se ha de construir. Se debe determinar la información que ha de usar el sistema, qué funciones debe realizar, qué condiciones existen, cuáles son las interfaces del sistema y qué criterios de validación se utilizarán.

El estudio y definición del problema dan lugar al planteamiento del problema que se escribirá en la documentación del programa. Si no se sabe lo que se busca, no se lo reconoce si se lo encuentra. Es decir que, si no sabemos con claridad qué es lo que tenemos que resolver, no podremos encontrar una solución. Aquí se declara cuál es la situación de partida y el entorno de datos de entrada, los resultados deseados, dónde deben registrarse y cuál será la situación final a la que debe conducir el problema después de ser implementado.

4.2 Fase de desarrollo

En esta fase se diseñan estructuras de datos y de los programas, se escriben y documentan éstos, y se prueba el software.

En esta etapa del ciclo de vida de desarrollo de programas, los analistas trabajan con los requerimientos del software desarrollados en la etapa de análisis. Se determinan todas las tareas que cada programa realiza, como así también, la forma en que se organizarán estas tareas cuando se codifique el programa. Los problemas cuando son complejos, se pueden resolver más eficientemente con el ordenador cuando se descomponen en subproblemas que sean más fáciles de solucionar que el original. 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 el ordenador se denomina diseño descendente (top-down design). 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.

En esta etapa además, se utilizan auxiliares de diseño, que son diagramas y tablas que facilitan la delineación de las tareas o pasos que seguirá el programa, por ejemplo: diagramas de flujo, pseudocódigo, etc.

En esta fase, se convierte el algoritmo en programa, escrito en un lenguaje de programación de alto nivel como C, Java, etc. La codificación del programa suele ser una tarea pesada que requiere un conocimiento completo de las características del lenguaje elegido para conseguir un programa eficaz. Sin embargo, si el diseño del algoritmo se ha realizado en detalle con acciones simples y con buena legibilidad, el proceso de codificación puede reducirse a una simple tarea mecánica. Las reglas de sintaxis que regulan la codificación variarán de un lenguaje a otro y el programador deberá conocer en profundidad dichas reglas para poder diseñar buenos programas.

Para aumentar la productividad, es necesario adoptar una serie de normas, como ser:

- Estructuras aceptables (programación estructurada)
- Convenciones de nominación: maneras uniformes de designación de archivos y variables.
- Convenciones de comentarios.

4.3 Fase de mantenimiento

Una vez obtenido el programa fuente, es necesaria su traducción al código máquina, ya que los programas escritos en un lenguaje de alto nivel no son directamente ejecutables por el ordenador. Según el tipo de traductor que se utilice, los lenguajes de alto nivel se clasifican en lenguajes interpretados y lenguajes compilados.

Son lenguajes interpretados aquellos en los que el sistema traduce una instrucción y la ejecuta, y así sucesivamente con las restantes.

Son lenguajes compilados aquellos en los que, primero se traduce el programa fuente completo, obteniéndose un código intermedio o módulo objeto (programa objeto); después, se fusiona éste con rutinas o librerías necesarias para su ejecución en un proceso llamado linkado y que obtiene como resultado un módulo ejecutable (programa ejecutable). La ventaja de los lenguajes compilados, frente a los interpretados, son su rápida ejecución y, en caso de necesitar posteriores ejecuciones del mismo programa, se hará del ejecutable almacenado.

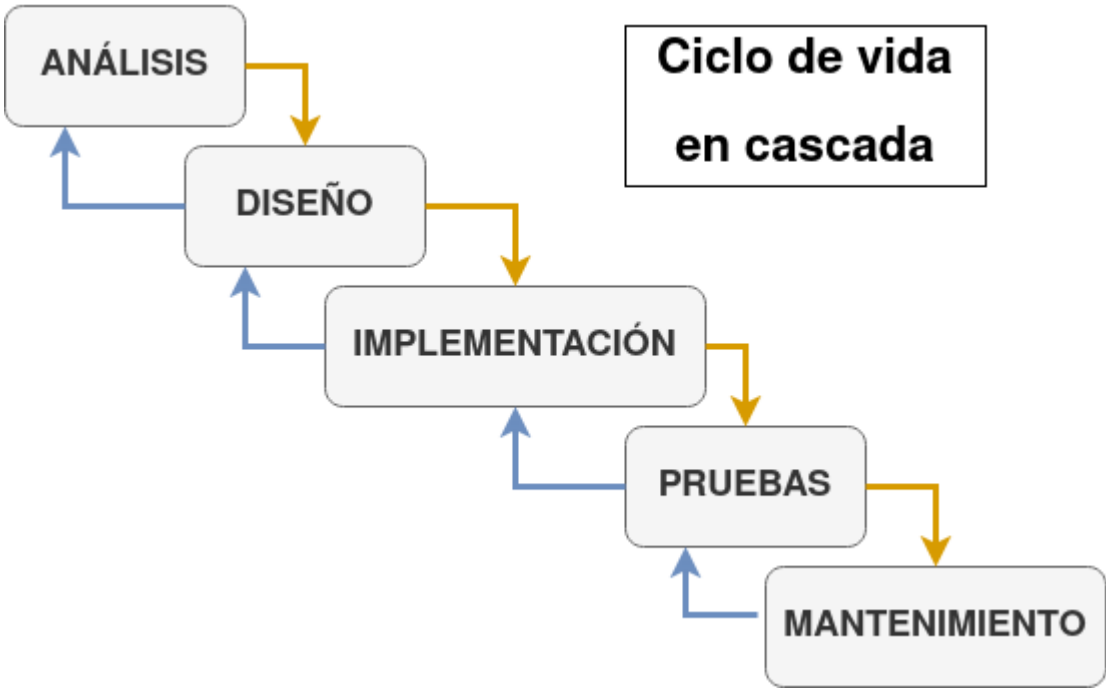
La puesta a punto consta de las siguientes etapas:

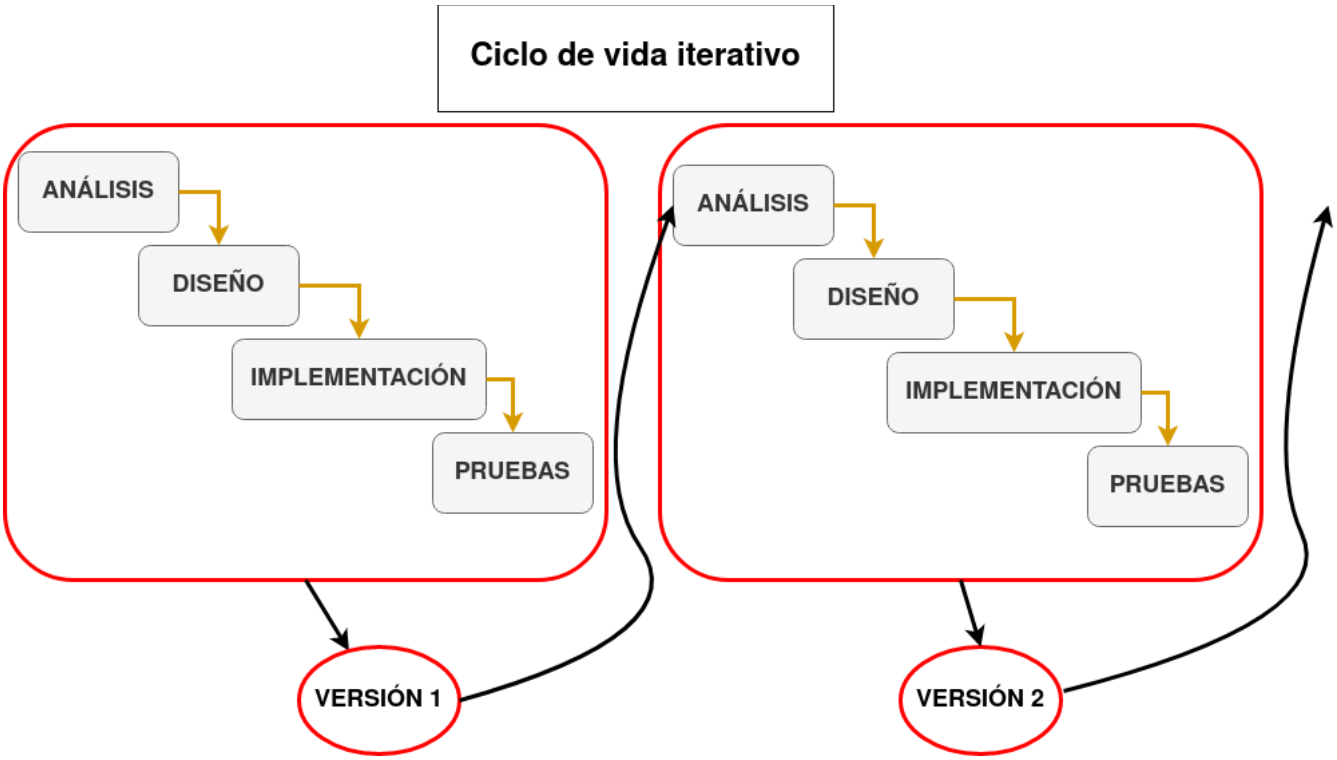
- Detección de errores.
- Depuración de errores.

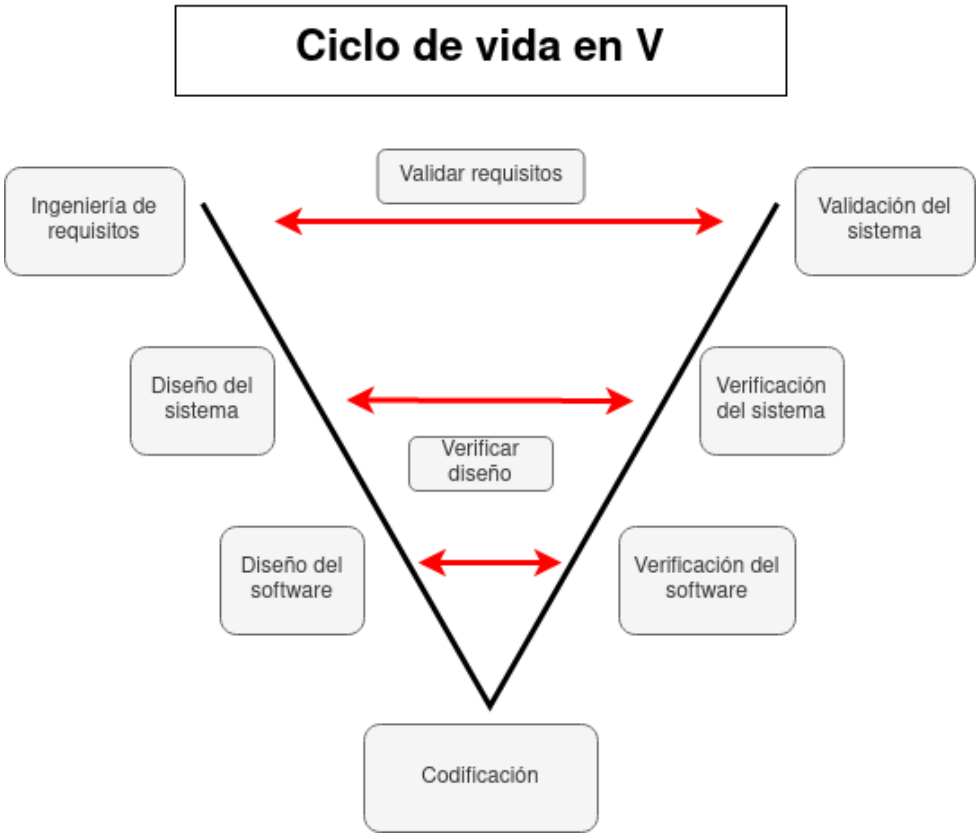
- Prueba del programa.

En cada una de estas fases se pueden detectar problemas que nos hacen replantearnos conceptos de la fase anterior y rehacer el software creado con las oportunas correcciones.

4.4 Tipos de ciclos de vida




Ciclo de vida iterativo
ANÁLISIS
DISEÑO
IMPLEMENTACIÓN
PRUEBAS
VERSIÓN 1
ANÁLISIS
DISEÑO
IMPLEMENTACIÓN
PRUEBAS
VERSIÓN 2


Ciclo de vida en V
Ingeniería de requisitos
Validar requisitos
Validación del sistema
Diseño del sistema
Verificar diseño
Verificación del sistema
Diseño del software
Verificación del software
Codificación

5 DOCUMENTACIÓN

La mayor parte de los proyectos exigen la realización de una planificación previa. Esta planificación debe determinar el modelo de ciclo de vida a seguir, los plazos para completar cada fase y los recursos necesarios en cada momento. Todo esto se debe plasmar en una documentación completa y detallada de toda la aplicación.

La documentación asociada al software puede clasificarse en interna y externa. La documentación interna corresponde a la que se incluye dentro del código fuente de los programas. Nos aclaran aspectos de las propias instrucciones del programa. La documentación externa es la que corresponde a todos los documentos relativos al diseño de la aplicación, a la descripción de la misma y sus módulos correspondientes, a los manuales de usuario y los manuales de mantenimiento.

> En el módulo que nos ocupa tendremos que documentar el código fuente que desarrollemos durante la elaboración de los distintos programas.

6 OBJETOS DE UN PROGRAMA

Entendemos por objeto de un programa todo aquello que pueda ser manipulado por las instrucciones. En ellos se almacenarán tanto los datos de entrada como los de salida (resultados).

Sus atributos son:

- Nombre: el identificador del objeto.
- Tipo: conjunto de valores que puede tomar.
- Valor: elemento del tipo que se le asigna.

6.1 Constantes

Son objetos cuyo valor permanece invariable a lo largo de la ejecución de un programa. Una constante es la denominación de un valor concreto, de tal forma que se utiliza su nombre cada vez que se necesita referenciarlo.

Por ejemplo: PI = *3.14.1592* *E = 2.718281*

También son utilizadas las constantes para facilitar la modificabilidad de los programas, es decir para hacer más independientes ciertos datos del programa. Por ejemplo, supongamos un programa en el que cada vez que se calcula un importe al que se debe sumar el IVA utilizáramos siempre el valor 0.16, en caso de variar este índice tendríamos que ir buscando a lo largo del programa y modificando dicho valor,

mientras que si le damos nombre y le asignamos un valor, podremos modificar dicho valor con mucha más facilidad.

6.2 Variables

Son objetos cuyo valor puede ser modificado a lo largo de la ejecución de un programa.

Por ejemplo: una variable para calcular el área de una circunferencia determinada, una variable para calcular una factura, etc.

6.3 Expresiones

Las expresiones según el resultado que produzcan se clasifican en:

- Numéricas: Son las que producen resultados de tipo numérico. Se construyen mediante los operadores aritméticos.

Por ejemplo:

```
pi * sqr(x)     (2*x)/3
```

- Alfanuméricas: Son las que producen resultados de tipo alfanumérico. Se construyen mediante operadores alfanuméricos.

Por ejemplo:

```
"Don " + "José"
```

- Booleanas o lógicas: Son las que producen resultados de tipo Verdadero o Falso. Se construyen mediante los operadores relacionales y lógicos.

Por ejemplo:

```
a < 0

(a > 1) and (b < 5)
```

6.4 Operadores

Son símbolos que hacen de enlace entre los argumentos de una expresión.

Relacionales

Se usan para formar expresiones que al ser evaluadas devuelven un valor booleano: verdadero o falso.

Operador	Definición
<	Menor que
>	Mayor que
==	Igual que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto que

Ejemplos:

Expresión	Resultado
A' < 'B'	Verdadero, ya que en código ASCII la A está antes que la B
1 < 6	Verdadero
10 < 2	Falso

📣 ASCII (acrónimo inglés de American Standard Code for Information Interchange — Código Estándar Estadounidense para el Intercambio de Información), es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno. Puedes ver la tabla en el siguiente enlace: http://ascii.cl/es/

Aritméticos

Se utilizan para realizar operaciones aritméticas.

Operador	Definición
+	Suma
-	Resta
*	Multiplicación
^	Potencia
/	División
%	Resto de la división

Ejemplos:

Expresión	Resultado
3 + 5 - 2	6
24 % 3	0
8 * 3 - 7 / 2	20.5

Lógicos o booleanos

La combinación de expresiones con estos operadores producen el resultado verdadero o falso.

Operador	Definición
No	Negación
Y	Conjunción
O	Disyunción

El comportamiento de un operador lógico se define mediante su correspondiente tabla de verdad, en ella se muestra el resultado que produce la aplicación de un determinado operador a uno o dos valores lógicos. Las operaciones lógicas más usuales son:

- NO lógico (NOT) o negación:

A	NOT A
V	F
F	V

El operador NOT invierte el valor: Si es verdadero (V) devuelve falso (F), y viceversa.

- O lógica (OR) o disyunción:

A	B	A OR B
V	V	V
V	F	V
F	V	V
F	F	F

El operador OR devuelve verdadero (V) si alguno de los dos valores es verdadero. De lo contrario, devuelve Falso (F).

- Y lógica (AND) o conjunción:

A	B	A AND B
V	V	V
V	F	F
F	V	F
F	F	F

El operador AND devuelve Verdadero (V) solo si ambos valores son verdaderos.

En cualquier otro caso devuelve Falso (F).Ejemplos (Suponiendo que a<b):

Expresión	Resultado
9 = (3*3)	Verdadero
3 <> 2	Verdadero
9 = (3*3) Y 3 <> 2	Verdadero
3 > 2 Y b < a	Verdadero Y Falso = Falso
3 > 2 O b < a	Verdadero O Falso = Verdadero
no(a < b)	No Verdadero = Falso
5 > 1 Y NO(b < a)	Verdadero Y no Falso = Verdadero

Paréntesis ()

Anidan expresiones.

Ejemplos:

Operación (3*2) + (6/2)	Resultado 9

Operador Alfanumérico (+)

Une datos de tipo alfanumérico. También llamado concatenación.

Ejemplos:

Expresión	Resultado
"Ana " + "López"	Ana López
"saca" + "puntas"	sacapuntas

Orden de evaluación de los operadores

A la hora de resolver una expresión, el orden a seguir es el siguiente:

1. Paréntesis.
2. Potencia ^
3. Multiplicación y división * /
4. Sumas y restas + -
5. Concatenación +
6. Relacionales < <= > >= etc.
7. Negación NOT
8. Conjunción AND
9. Disyunción OR

La evaluación de operadores de igual orden se realiza de izquierda a derecha. Este orden de evaluación tiene algunas modificaciones en determinados lenguajes de programación.

7 REPRESENTACIÓN

Existen diversas formas de representación de algoritmos. Las más importantes son los diagramas de flujo (también llamados 'ordinogramas') y el pseudocódigo.

7.1 Diagramas de flujo (ordinogramas)

Durante el diseño de un programa y en sus fases de análisis y programación, surge la necesidad de representar de una manera gráfica los flujos que van a seguir los datos manipulados por el mismo, así como la secuencia lógica de las operaciones para la resolución del problema.

Esta representación gráfica debe tener las siguientes cualidades:

1. Sencillez en su construcción.
2. Claridad en su compresión.
3. Normalización en su diseño.
4. Flexibilidad en sus modificaciones.

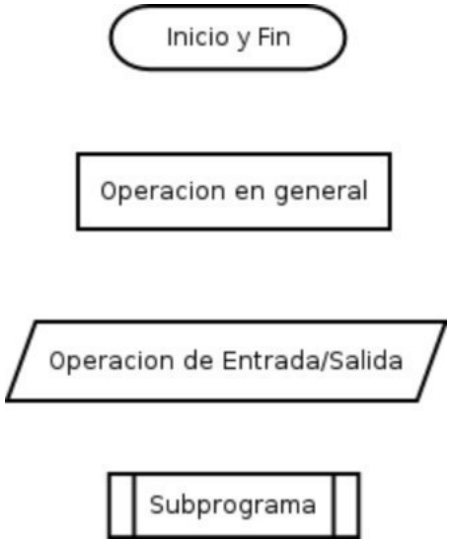
> En la práctica se suelen utilizar indistintamente los términos diagrama de flujo, organigrama y ordinograma para referenciar cualquier representación gráfica de los flujos de datos o de las operaciones de un programa.
> Es importante diferenciarlos porque no corresponden a las mismas fases de diseño de los programas. Aunque utilicen algunos símbolos comunes, el significado de éstos no es el mismo.

En la representación de ordinogramas es conveniente seguir las siguientes reglas:

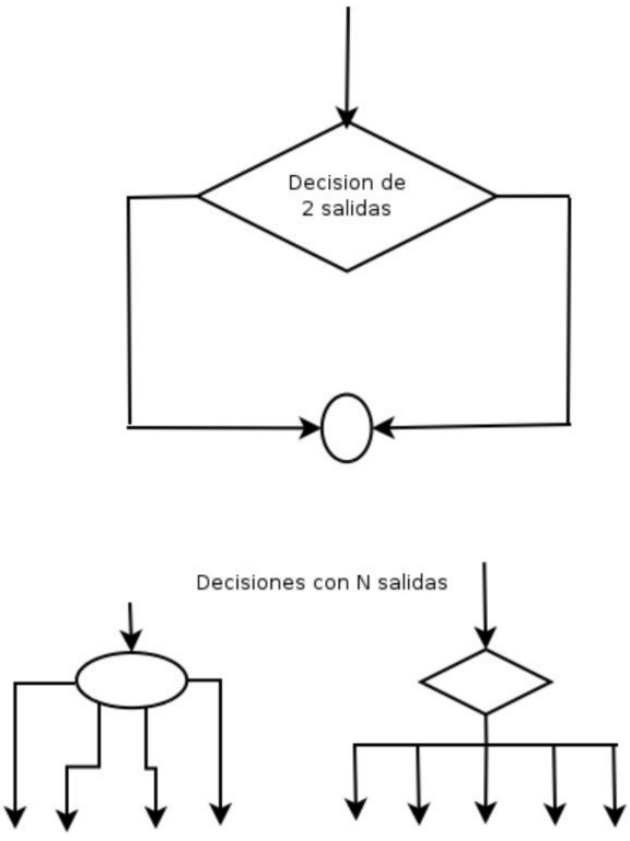
- El comienzo del programa figurará en la parte superior del ordinograma.
- El símbolo de comienzo deberá aparecer una sola vez en el ordinograma.
- El flujo de las operaciones será, siempre que sea posible de arriba a abajo y de izquierda a derecha.
- Se evitarán siempre los cruces de líneas utilizando conectores.

Esta será la representación que utilizaremos durante el curso.

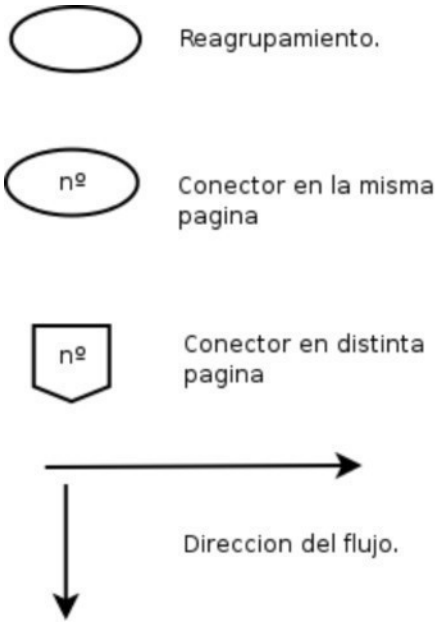
Símbolos de operación



Símbolos de decisión



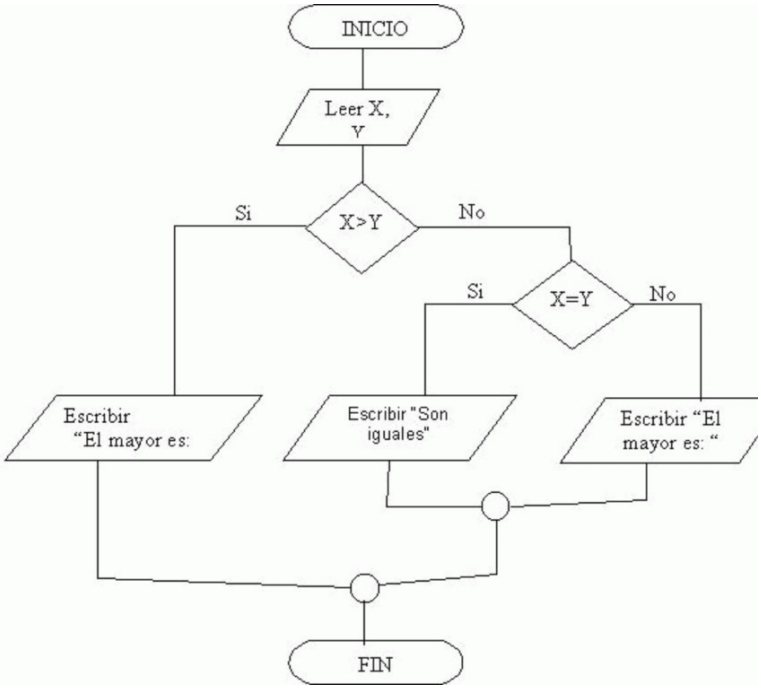
Símbolos de conexión



Ejemplo

Algoritmo que lee dos números “X” e “Y”, determina si son iguales, y en caso de no serlo, indica cuál de ellos es el mayor.

Su representación gráfica mediante ordinograma podemos verla en el gráfico:



7.2 Pseudocódigo

Además de las representaciones gráficas, un programa puede describirse mediante un lenguaje intermedio entre el lenguaje natural y el lenguaje de programación, de tal manera que permita flexibilidad para expresar las acciones que se van a realizar y, también imponga algunas limitaciones, que tienen importancia cuando se quiere codificar el programa a un lenguaje de programación determinado.

La notación en pseudocódigo se caracteriza por:

a) Facilitar la obtención de la solución mediante la utilización del diseño descendente o Topdown.

b) Ser una forma de codificar los programas o algoritmos fácil de aprender y utilizar.

c) Posibilitar el diseño y desarrollar los algoritmos de una manera independiente del lenguaje de programación que se vaya a utilizar cuando se implemente el programa.

d) Facilitar la traducción del algoritmo a un lenguaje de programación específico.

e) Permitir un gran flexibilidad en el diseño del algoritmo que soluciona el problema, ya que se pueden representar las acciones de una manera mas abstracta, no estando sometidas a las reglas tan rígidas que impone un lenguaje de programación.

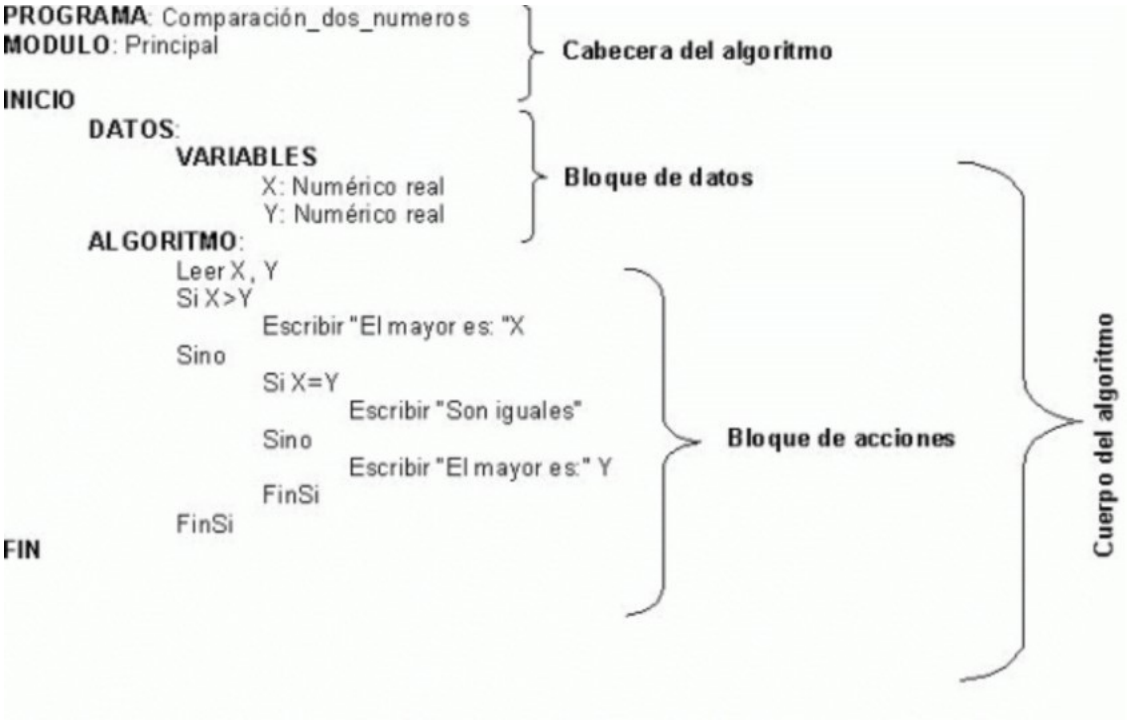
f) Posibilitar futuras correcciones y actualizaciones en el diseño del algoritmo por la utilización una serie de normas, que acotan el trabajo del desarrollador.

Cuando se escribe un algoritmo mediante la utilización de pseudocódigo, se debe "sangrar" el texto con respecto al margen izquierdo, con la finalidad de que se comprenda más fácilmente el diseño que se está realizando.

Ejemplo

Algoritmo que lee dos números “X” e “Y”, determina si son iguales, y en caso de no serlo, indica cuál de ellos es el mayor.

Su representación gráfica mediante pseudocódigo será:


PROGRAMA: Comparación_dos_numeros
MODULO: Principal
INICIO
DATOS:
VARIABLES
X: Numérico real
Y: Numérico real
ALGORITMO:
Leer X, Y
Si X>Y
Escribir "El mayor es: "X
Sino
Si X=Y
Escribir "Son iguales"
Sino
Escribir "El mayor es:" Y
FinSi
FinSi
FIN
Cabecera del algoritmo
Bloque de datos
Bloque de acciones
Cuerpo del algoritmo