

1. Introducción a la programación.

Desarrollo de Aplicaciones Multiplataforma. Programación.

Contenido

1. Qué es la programación.

- 1.1 Introducción.
- 1.2 Fases de la programación.
- 1.3 Lenguajes de programación.
- 1.4 Entornos integrados de desarrollo.

2. Algoritmos.

- 2.1 Definición.
- 2.2 Representación de algoritmos.
 - 2.2.1 Diagramas de flujo.
 - Organigramas.
 - Ordinogramas.
 - 2.2.2 Pseudocódigo.

3. Metodología de la programación.

- 3.1 Concepto de programa.
- 3.2 Estructura de un programa.
- 3.3 Elementos de un programa.
 - 3.3.1 Constantes y variables.
 - 3.3.2 Tipos de datos: Simples y estructurados.
 - 3.3.3 Operadores y expresiones.
 - Operadores.
 - Funciones internas.
 - Expresiones.
 - 3.3.4 Instrucciones.
 - Instrucción de asignación.
 - Instrucciones de control.
 - Subprogramas.
- 3.4 Variables auxiliares.
 - 3.4.1 Contadores.
 - 3.4.2 Acumuladores.
 - 3.4.3 Switches.

1. Qué es la programación.

1.1 Introducción.

- Todos estamos familiarizados con el uso de programas y la ejecución de los mismos: editores de textos, navegadores, juegos, reproductores de música o películas, etc.
- Por regla general, cuando queremos ejecutar un programa se lo indicamos al sistema haciendo doble click sobre él o incluso los usuarios avanzados ejecutan comandos desde una consola para conseguir una función determinada.
- La finalidad de este módulo es aprender a programar; crear programas, colecciones de instrucciones que lleven a cabo una función concreta, ejecuten una serie de órdenes para llevar a cabo aplicaciones informáticas, crear software más o menos complejo.
- En esta unidad en concreto nos acercaremos a la programación, sus fases, el lenguaje en el que trabajaremos y el entorno de desarrollo que utilizaremos para “hablar” ese lenguaje. Será importante distinguir conceptos como algoritmo y programa, y manejar el vocabulario y los conceptos básicos para un programador.

1. Qué es la programación.

1.2 Fases de la programación.

La elaboración de un programa tiene su origen en la necesidad de realizar una tarea de forma automatizada, es decir rutinaria, siguiendo unos pasos concretos y mecánicos y que informatizados van a ahorrar tiempo a quien los debe ejecutar. El ordenador va a ser la herramienta que resuelva estas situaciones, pero la creación de un programa ha de seguir unas fases determinadas para que sea realmente eficaz, así que su creación hay que abordarla, como muchas de las situaciones cotidianas, con un orden y un rigor.

Creación de una rutina...
Observación de la situación o problema.
Pasos para solucionarlo.
Describir una o varias posibles soluciones.
Aplicar la solución que estimamos más adecuada.
Comprobar si esa solución recoge todas las situaciones que se pueden dar.

Ya sea en la vida real o en la programación, a la hora de automatizar una determinada tarea, debemos dar solución a un problema dando una serie de pasos que deben estar bien definidos. El proceso de creación de software puede dividirse en diferentes fases:

- Fase de planteamiento y análisis del problema.
- Fase de implementación.
- Fase de explotación y mantenimiento.

1. Qué es la programación.

Fase de planteamiento y análisis del problema.

a. Planificación:

- En esta fase es necesario conocer por encima la actividad que se quiere informatizar y plantear varias soluciones informáticas.

b. Análisis:

- Consiste en conocer a fondo la actividad a informatizar.
- El análisis indica los aspectos concretos que se deben abordar. Debe existir una buena comunicación y un alto grado de comprensión entre el analista/programador y el cliente/usuario, para así conocer las necesidades que precisa la aplicación. Se especificarán los procesos y estructuras de datos que se van a emplear. En esta fase se analiza la documentación de la empresa, se investiga, y observa todo lo que rodea el problema y se recopila cualquier información útil.

c. Diseño del programa:

- Se diseñan todas las entradas y salidas que va a tener el programa.
- Se indican todos los pasos que tiene que realizar un programa con el fin de realizar una determinada tarea: algoritmo.
- Una vez creado el algoritmo se comprueba con datos concretos, y se estudian con detalle aquellas situaciones extremas en las que el algoritmo puede fallar o resultar incompleto y en tal caso se retoma el diseño del programa.

1. Qué es la programación.

Fase de implementación.

a. Codificación:

- Consiste en transformar o traducir los resultados obtenidos a un determinado lenguaje de programación; es decir transformar el algoritmo en programa.
- Hay que elegir un lenguaje de programación, tener en cuenta las reglas de este lenguaje, buscar un entorno de desarrollo que nos facilite el trabajo, y obtener así el programa.
- Pero para que nuestro programa comience a funcionar, antes debe ser traducido a un lenguaje que la máquina entienda. Este proceso de traducción puede hacerse de dos formas, compilando o interpretando el código del programa.
- Una vez traducido, sea a través de un proceso de compilación o de interpretación, el programa podrá ser ejecutado.

1. Qué es la programación.

Fase de implementación.

b. Ejecución y validación:

- En esta fase se implanta el programa en el sistema donde es necesario, se ejecuta y se comprueba su funcionamiento. De nuevo se prueban casos límites en busca de posibles fallos y se registra un porcentaje de error para plantear mejoras.

c. Documentación:

- De cara a facilitar la ejecución, explotación y mantenimiento del programa, de cara a facilitar el trabajo a otros e incluso al propio programador, a la interpretación del código en posteriores consultas, a la ampliación del programa o cambios necesarios en un futuro,...el programa deberá ir acompañado de documentación.
- Esta puede ser interna: encabezados, descripciones, declaraciones del problema y comentarios, y documentación externa: manuales que se crean para una mejor ejecución y utilización del programa.

1. Qué es la programación.

Fase de mantenimiento del programa.

- El mantenimiento del programa consiste en la evaluación periódica del mismo una vez que ya ha sido instalado y entregado al cliente. En esta fase de explotación se pueden hacer mejoras y adaptaciones y correcciones de errores detectados durante su uso.

1. Qué es la programación.

1.3 Lenguajes de programación.

- Un lenguaje de programación es un lenguaje diseñado para hacer llegar al ordenador las diferentes tareas que tiene que realizar, y así poder llevar a cabo una determinada actividad.
- Los lenguajes de programación pueden ser clasificados en función de lo cerca que estén del lenguaje humano o del lenguaje de los computadores. El lenguaje de los computadores son códigos binarios, es decir, secuencias de unos y ceros, y se le denomina lenguaje máquina.
- Como este lenguaje no es fácil de utilizar para el programador, se han desarrollado lenguajes más fáciles de manejar que siempre van acompañados de un programa traductor que traduce el lenguaje a lenguaje máquina.

Los lenguajes de programación se pueden clasificar atendiendo a varios criterios y las principales clasificaciones son:

- Según el nivel de abstracción.
- Según la forma de ejecución.

1. Qué es la programación.

Según el nivel de abstracción

- Lenguaje de bajo nivel: son los lenguajes de máquinas, aprovechan al máximo los recursos del ordenador donde se van a ejecutar. A este le sigue el lenguaje ensamblador de alta dificultad en su manejo.
- Lenguaje de alto nivel: traducen los algoritmos a un lenguaje más sencillo para el programador y no exclusivo de la máquina en la que se ejecuta.

1. Qué es la programación.

Según la forma de ejecución

- Lenguajes compilados: se traducen a través del programa compilador, que pasa todo el código a lenguaje máquina, generando un nuevo fichero ejecutable que contiene la misma información que el fichero original (código fuente) pero escrito en lenguaje máquina.

Si no hay cambios en el fichero original, la traducción se hace una sola vez, y no hará falta volverla a hacer. Ejemplos de este lenguaje son: C, Pascal,...

- Lenguajes interpretados: traducen el programa cada vez que se quiere ejecutar a través del intérprete. Estos no generan un fichero ejecutable, pero sí ejecutan cada línea de programa aunque el programa contenga algún error. Al detectar un error salta el programa, por lo que es más sencillo localizar errores y subsanarlos. Un ejemplo de este programa es Basic.
- Lenguajes intermediarios: están diseñados para obtener las ventajas de ambos tipos de lenguajes y combinan las dos tareas:

Primero se realiza la fase de compilación, traduciendo el código fuente a un lenguaje intermedio. Dicha traducción se guarda en un fichero.

En segundo lugar dicho fichero es traducido al lenguaje máquina cada vez que se quiera ejecutar (fase de interpretación).

Un ejemplo de este último es el lenguaje Java. En Java los ficheros que contienen el código fuente tienen extensión .java, y los ficheros que contienen la traducción al lenguaje intermedio son de extensión .class. El lenguaje intermedio que usa Java se llama "bytecodes".

1. Qué es la programación.

1.4 Entornos integrados de desarrollo.

- Son aplicaciones que ofrecen la posibilidad de llevar a cabo el proceso completo de desarrollo de software a través de un único programa.
- Podremos realizar las labores de edición, compilación, depuración, detección de errores, corrección y ejecución de programas escritos en Java o en otros lenguajes de programación, bajo un entorno gráfico (no mediante línea de comandos).
- Junto a las capacidades descritas, cada entorno añade otras que ayudan a realizar el proceso de programación, como por ejemplo: código fuente coloreado, plantillas para diferentes tipos de aplicaciones, creación de proyectos, etc.
- Este programa es una herramienta de desarrollo de programas, conocida como IDE (Integrated Development Environment).
- Para el lenguaje de programación Java existen múltiples alternativas, siendo los principales entornos de desarrollo NetBeans, Eclipse y JCreator. Los dos primeros son gratuitos, con soporte de idiomas y multiplataforma (Windows, Linux, MacOS).

2. Algoritmos.

2.1 Definición.

- Cómo vimos en la “Fase de Planteamiento y Análisis del problema” (apartado 1.2), un algoritmo es la pieza clave del diseño del programa. Se define como un conjunto ordenado y finito de reglas u órdenes que ejecutadas tal y como se escriben resuelven el problema planteado con independencia de los datos o circunstancias del problema. Está permitido que el desarrollo del algoritmo se realice en el lenguaje habitual del programador.
- El algoritmo no debe tener ambigüedades, y por tanto debe contemplar todas y cada una de las situaciones diferentes que puedan presentarse en distintas resoluciones del problema.

2. Algoritmos.

La resolución de un problema puede hacerse mediante diferentes algoritmos. Pero no todos tienen la misma calidad. Los parámetros para medir la calidad de los algoritmos son:

- Mínima ocupación de memoria.
- Mínimo tiempo de ejecución.
- Legibilidad, es decir, debe estar escrito de forma que se facilite su lectura y comprensión.
- Portabilidad, es decir el algoritmo debe estar concebido de forma que se pueda codificar con facilidad en distintos lenguajes de programación.
- Modificabilidad, rara vez un programa no debe adaptarse a nuevas exigencias del usuario, por ello los programas deben estar escritos de forma que contemplen esta circunstancia.

2. Algoritmos.

En conclusión, todo algoritmo debe tener las siguientes características:

- Un punto de inicio.
- Estar bien definido, es decir hacer lo que se desea que haga. Si se parte de los mismos datos, siempre se ha de obtener el mismo resultado.
- Tener en cuenta todas las posibles situaciones que puedan plantearse.
- Ser preciso indicando el orden de realización de cada uno de sus pasos.
- Fácil de leer e interpretar.
- Ser finito. Al seguir un algoritmo este debe concluir en algún momento. Tiene un número finito de pasos.
- Terminar en un solo punto.
- Estar diseñado de forma que sea independiente del lenguaje de programación en que vaya a ser codificado.

Cuando el algoritmo cumpla las características anteriores, se podrá escribir el programa. A diferencia del algoritmo, un programa debe estar escrito en un lenguaje concreto de programación con sus instrucciones concretas y su estructura específica, como veremos más adelante.

2. Algoritmos.

2.2 Representación de algoritmos.

- A la hora de representar el algoritmo hay que utilizar técnicas que permitan independizar dicho algoritmo del lenguaje de programación en el que, posteriormente, codifiquemos el programa correspondiente. Es decir, que la representación del algoritmo, por un lado, no debe depender de la sintaxis del lenguaje de programación que vayamos a utilizar posteriormente, pero por otro, debe ser fácilmente transformable a un programa escrito en el lenguaje de programación que posteriormente elijamos.
- Para describir algoritmos se utilizan técnicas de representación que veremos a continuación. Las representaciones que obtengamos aplicando cada una de estas técnicas serán la base para realizar el programa en la fase de codificación, por ello estas representaciones deben ser amplias, ordenadas, fáciles de entender y posibles de modificar.

Las representaciones más comunes son:

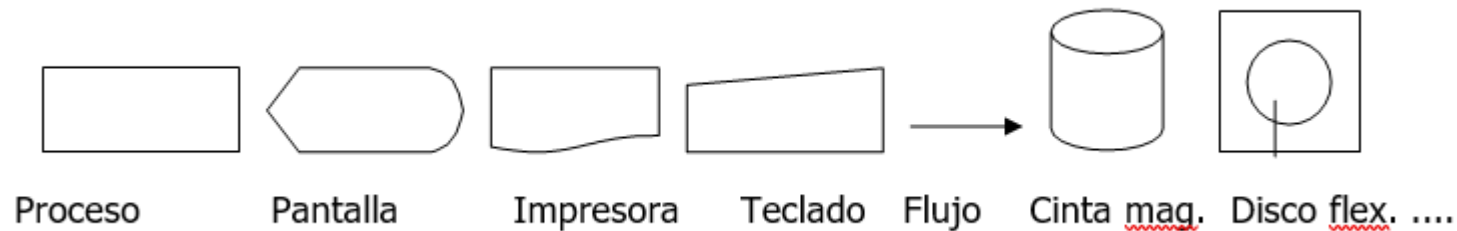
- Los diagramas de flujo del sistema u organigramas.
- Los diagramas de flujo de proceso u ordinogramas.
- El pseudocódigo.

2. Algoritmos.

2.2.1 Diagramas de flujo.

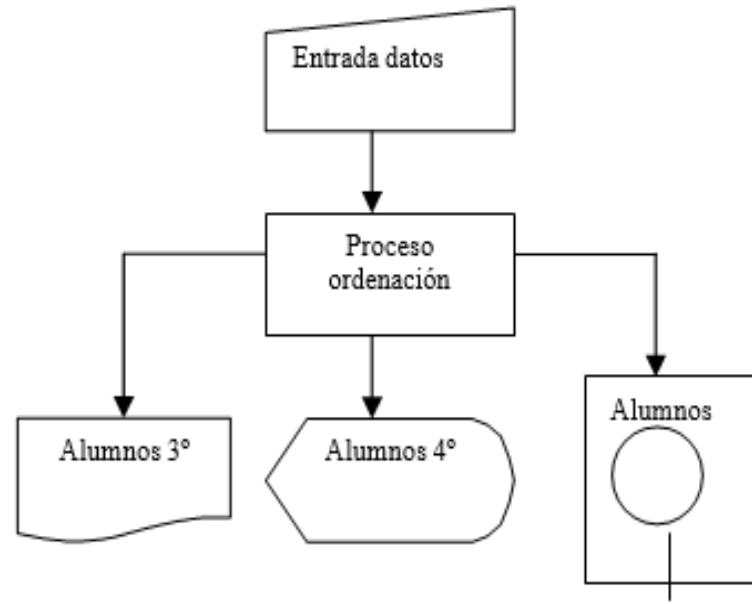
- Organigramas

Son representaciones que describen el flujo de datos y los soportes físicos que intervienen en la solución del problema. Se trata de una representación escueta que da una imagen global de lo que se quiere hacer, sin entrar en detalles. Es la representación más sencilla y usa varios símbolos:



2. Algoritmos.

Ejemplo: los alumnos de un centro han de distribuirse en grupos y se introduce el listado por teclado; cada grupo formado se guardará en un disco flexible. Se sacará un listado de los alumnos de 4º curso por pantalla y los de 3º por impresora.



2. Algoritmos.

- Ordinogramas

Los ordinogramas son herramientas gráficas para representar algoritmos que muestran de forma detallada y lógica las operaciones que se van a realizar cuando se haga un seguimiento del algoritmo (o la ejecución del programa). Aunque fueron muy utilizados en los principios de la programación (programación convencional), con la aparición de la programación estructurada están en desuso.


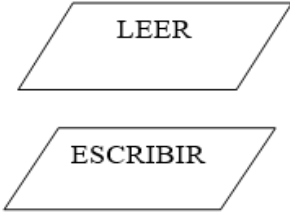
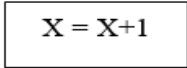

Un ordinograma está compuesto por una serie de símbolos unidos por flechas. Los símbolos, representan acciones y las flechas el orden de realización de las acciones. Cada símbolo por tanto tendrá al menos una flecha que conduzca a él y otra que parta de él.

Los principales símbolos que se utilizan en un ordinograma se pueden agrupar en las siguientes categorías:

- a) Símbolos de proceso.
- b) Símbolos de decisión.
- c) Líneas de flujo.
- d) Símbolos de conexión.
- e) Símbolos de comentarios.

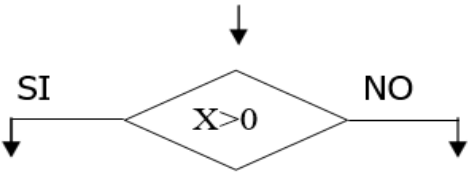
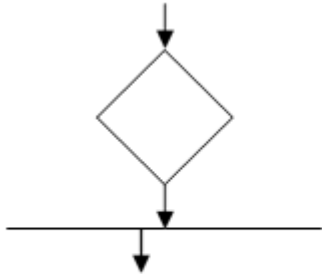
2. Algoritmos.

a) Símbolos de proceso

SÍMBOLOS	FUNCIÓN
 <p>Diagram showing two terminal symbols: a rounded rectangle labeled "INICIO" and another rounded rectangle labeled "FIN".</p>	Terminal o terminador: Representa el "INICIO" y el "FIN" de un algoritmo.
 <p>Diagram showing two input/output symbols: a parallelogram labeled "LEER" and another parallelogram labeled "ESCRIBIR".</p>	Operación de entrada /salida Utilizada para mostrar la INTRODUCCIÓN o LECTURA de datos desde un periférico a la memoria del ordenador y la salida, VISUALIZACIÓN o ESCRITURA de resultados desde la memoria del ordenador a un periférico.
 <p>Diagram showing a process symbol: a rectangle labeled "X = X+1".</p>	Proceso: Utilizado para mostrar una operación que pueda originar cambio de valor, operaciones aritméticas, etc.
 <p>Diagram showing a subprograma symbol: a rectangle with double vertical lines on the left and right sides.</p>	Subprograma: Se utiliza para representar la llamada a una función o procedimiento que va a llevar a cabo una tarea determinada y una vez terminada ésta, regresar el control de ejecución al programa principal.

2. Algoritmos.

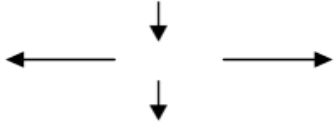
b) Símbolos de decisión

SIMBOLOS	FUNCIÓN
	Decisión: Representa operaciones lógicas o de comparaciones entre datos y en función del resultado de la misma, determina cuál de los distintos caminos alternativos del algoritmo se debe seguir. Normalmente tiene dos salidas SI y NO .
	Decisión Múltiple: Utilizada para representar una decisión con tres o más salidas.



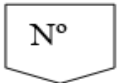
b) Símbolos de decisión

2. Algoritmos.

c) Líneas de flujo

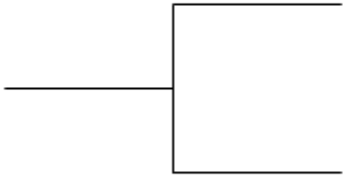
SIMBOLOS	FUNCIÓN
	Líneas de flujo o conectoras: Indican el sentido de la ejecución de las operaciones. Sirven de unión entre dos símbolos.

d) Símbolos de conexión

SIMBOLOS	FUNCIÓN
	Conector de reagrupamiento : Utilizado para el reagrupamiento de líneas de flujo
	Conector de una misma página: Sirve para conectar dos puntos o partes del ordinograma en la misma página.
	Conector de distintas páginas: Sirve para conectar dos puntos del ordinograma en páginas diferentes.

2. Algoritmos.

e) Símbolos de comentarios

SIMBOLOS	FUNCIÓN
	Permite escribir comentarios a lo largo del diseño realizado

2. Algoritmos.

En los ordinogramas sólo queda representado gráficamente el bloque de instrucciones, siendo necesario especificar explícitamente el entorno del algoritmo.

- A la hora de obtener la representación gráfica del bloque de instrucciones de un algoritmo hay que tener en cuenta que:
- Todo ordinograma debe indicar claramente dónde comienza (INICIO) y dónde acaba (FIN). Existe un único "INICIO" y un único "FIN".
- El orden en que deben escribirse los símbolos es de arriba a abajo y de izquierda a derecha.
- Dentro de los símbolos no se especificarán instrucciones de un determinado lenguaje de programación.
- Se puede escribir más de un paso del algoritmo en un símbolo de proceso, aunque es aconsejable usar un símbolo para cada acción.

2. Algoritmos.

- La secuencia de ejecución de los diferentes tratamientos se indica mediante líneas de conexión (horizontales/verticales), que los comunica entre sí. Estas líneas deben acabar en puntas de flecha que indiquen el flujo.
- A todos los símbolos excepto al de INICIO, les debe llegar una línea de conexión y sólo una.
- De todos los símbolos, excepto el de FIN y el de DECISIÓN, debe salir una sola línea de conexión.
- Para eliminar líneas de conexión muy largas se usan los conectores. Debe tenerse en cuenta, sin embargo, que el uso excesivo de éstos dificulta el entendimiento del ordinograma.
- No está permitido el cruce de líneas de flujo.
- El ordinograma en conjunto debe guardar una cierta simetría.

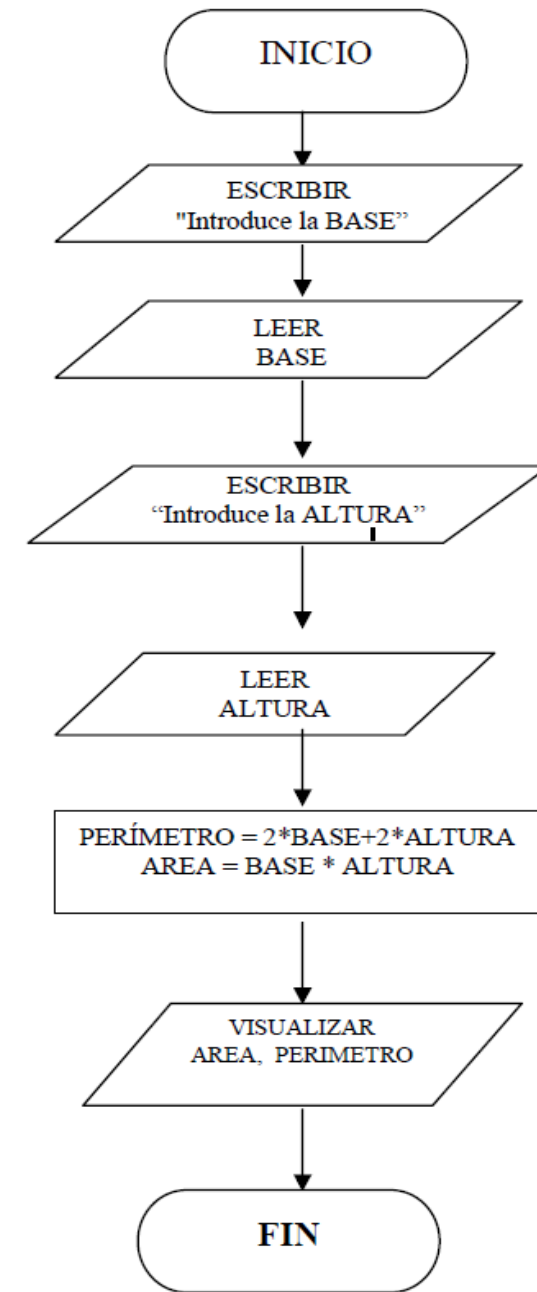
Ejemplo 1: Ordinograma del algoritmo que calcula el perímetro y el área de un rectángulo.

Algoritmo REC_AREA_PERIMETRO

Entorno

Entero BASE, ALTURA, PERIMETRO, AREA;

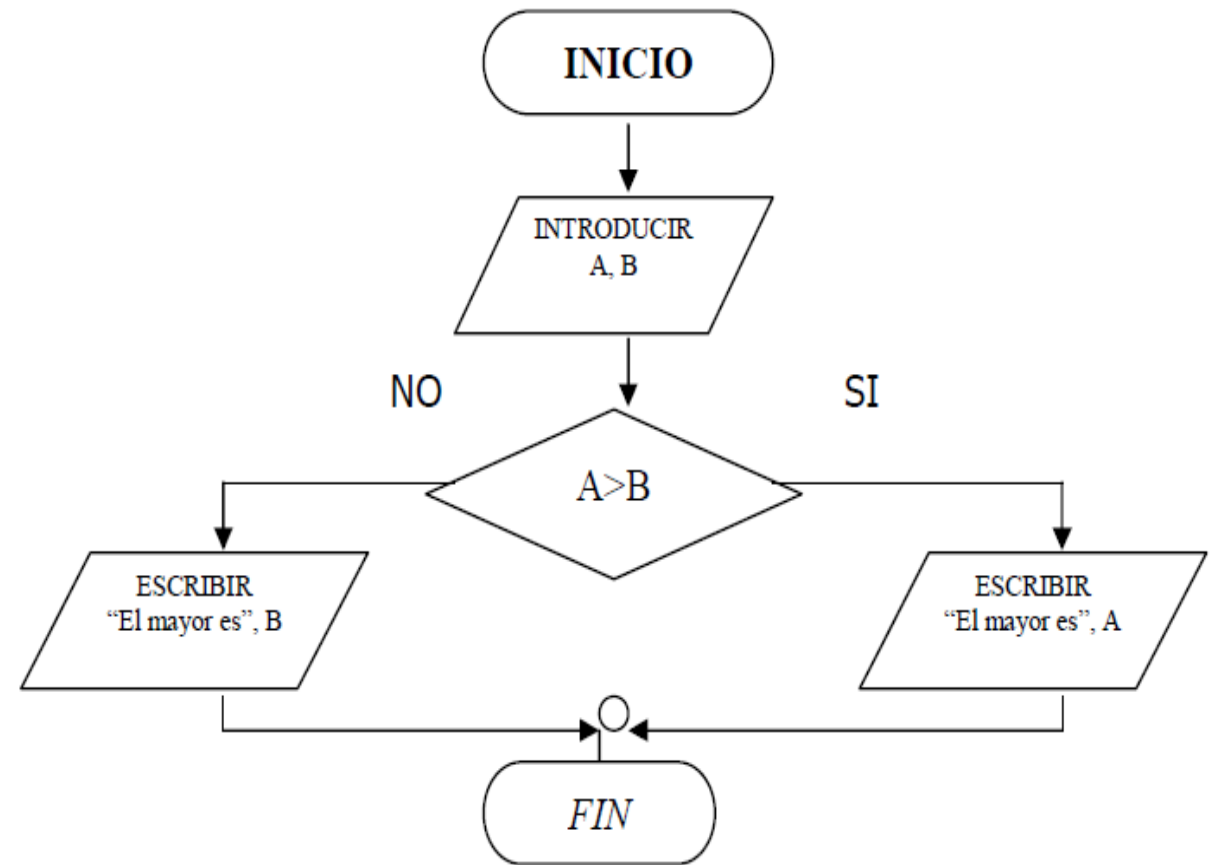
Fin_entorno



Ejemplo 2: Introducir dos números y visualizar el mayor.

Entorno Real A, B;

Fin_entorno



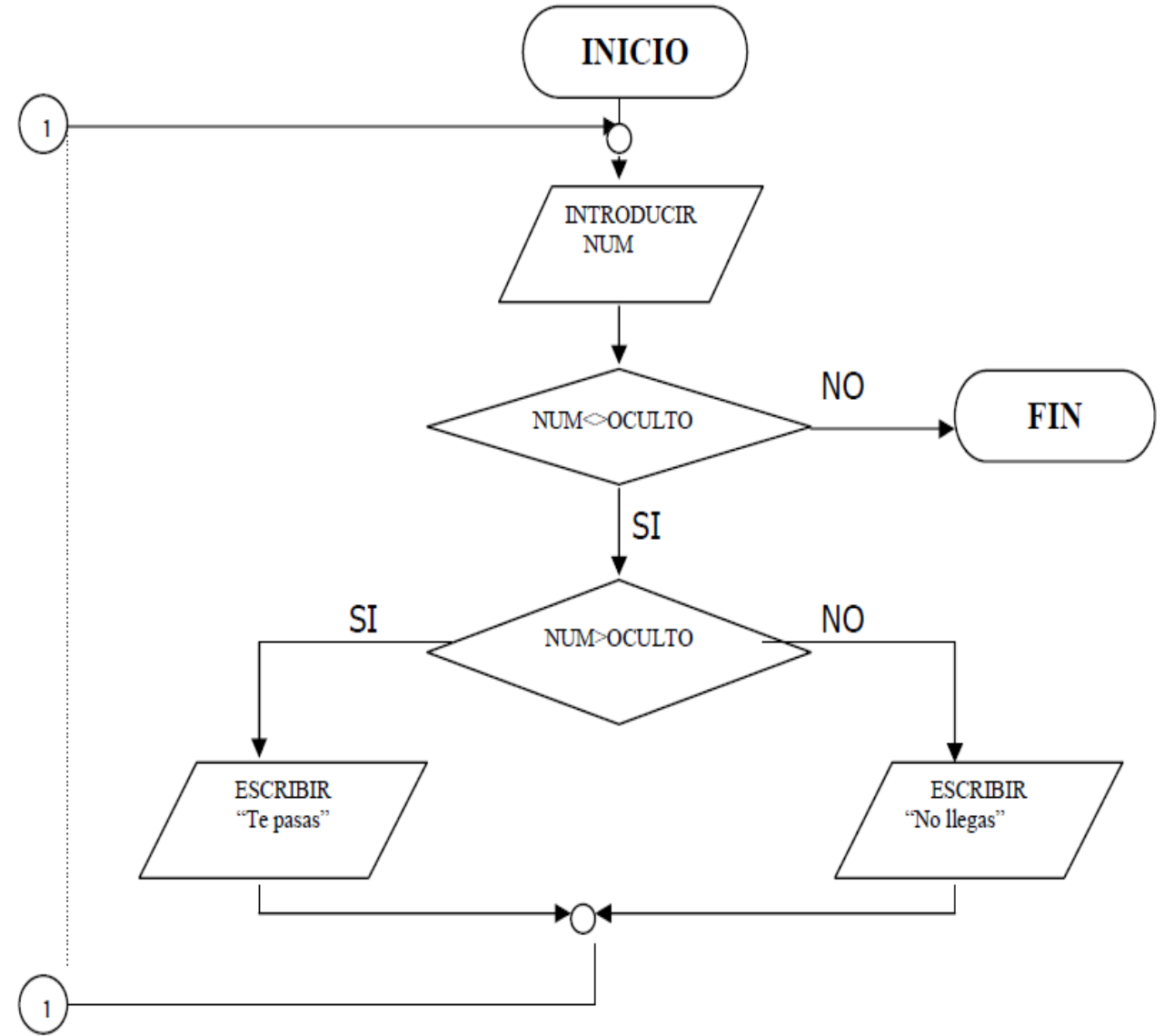
Ejemplo 3: Introducir un número por teclado, si no se adivina el número oculto, borrar pantalla, indicar si el n° es superior o inferior. Repetir la operación hasta acertar.

Entorno

Entero NUM;

Constante OCULTO=19

Fin_entorno



2. Algoritmos.

2.2.1 Pseudocódigo.

- El pseudocódigo es una técnica de representación de algoritmos no gráfica que nos permite describirlos mediante un lenguaje intermedio entre el lenguaje natural que normalmente utilizamos en nuestra comunicación escrita (el español) y el lenguaje de programación que posteriormente vamos a utilizar (Java).
- El que sea un lenguaje intermedio entre el lenguaje natural y el de programación es porque el pseudocódigo permite escribir la solución de un problema en forma de algoritmo utilizando palabras y frases del lenguaje natural sujetas a unas determinadas reglas que luego facilitan la traducción del algoritmo a un programa escrito en el lenguaje de programación elegido.

2. Algoritmos.

La utilización de pseudocódigo para describir algoritmos además de facilitar la traducción del algoritmo a un programa escrito en el lenguaje de programación elegido tiene las siguientes ventajas:

- Hace que, en la fase de diseño, los programadores se centren en la lógica y estructuras de control del algoritmo (descripción de las secuencias de pasos que hay que llevar a cabo) y se olviden de las reglas y restricciones sintácticas (como hay que escribir dicha secuencia) que le impone un determinado lenguaje de programación.
- La descripción o representación de los algoritmos que obtenemos es más fácil de crear y de entender, pues está realizada en el lenguaje que utilizamos habitualmente, no siendo necesario por tanto el conocimiento de un lenguaje de programación.
- La descripción o representación de los algoritmos que obtenemos es totalmente independiente de lenguaje de programación que posteriormente vayamos a utilizar.
- Facilita la realización de futuras correcciones o actualizaciones gracias a que no es un sistema de representación rígido.

2. Algoritmos.

La descripción del algoritmo que obtenemos mediante pseudocódigo, se considera como un primer borrador del programa que vamos a desarrollar en la fase de codificación, ya que como ya hemos dicho, la descripción obtenida en pseudocódigo es fácilmente traducible a un programa.

Es aconsejable respetar las siguientes reglas de carácter general:

- Todo pseudocódigo tiene un INICIO y un FIN.
- Cada instrucción debe escribirse en una línea.
- Para su descripción se usan palabras reservadas, que en un principio iban en inglés: if, for, while, etc, pero que actualmente se escriben en el lenguaje de cada país: si, para, mientras, etc....
- Debe escribirse indentado (con tabulaciones y sangrías) para mostrar claramente las dependencias de control dentro de los módulos.
- Cada estructura usada tendrá un solo punto de comienzo y un solo punto de fin de estructura. Algunos autores suelen usar un corchete para unir el principio y fin de cada estructura.
- Se escribirá en minúscula, excepto aquellos nombres que elige el programador: variables, ficheros, módulos, etc., que se pueden escribir en mayúsculas.
- Para referenciar un módulo se especifica simplemente su nombre entre los símbolos "<" y ">".

2. Algoritmos.

La representación de un algoritmo mediante pseudocódigo queda dividida en dos partes:

- **Cabecera:** Es el área o bloque informativo donde quedará reflejado el nombre del algoritmo. Incluirá además un comentario que nos dé una idea de lo que se pretende que resuelva el algoritmo. Los comentarios en un pseudocódigo quedan representados mediante:

`// si el comentario ocupa una sola línea.`

`/*.....*/ si el comentario ocupa varias líneas (en este caso el comentario iría entre estos dos caracteres).`

- **Cuerpo:** Es la zona donde se describen las partes de un algoritmo Entorno y Bloque de instrucciones. Cada una de estas partes a su vez suele estar dividida en otras, lo que exige la utilización de indentación o sangrado para que cada una de ellas quede fácilmente identificada y comprendida.

Ejemplo esquematizado de las partes en que se divide la representación:

Cabecera

Algoritmo DESCUENTO

/* Este algoritmo calcula el tanto por ciento de descuento que han hecho al realizar una determinada compra*/

Cuerpo

Entorno

/* En este bloque irá la declaración de los distintos elementos u objetos que vamos a utilizar en un algoritmo: tipos definidos por el usuario variables, constantes, funciones y procedimientos, etc.*/

Entero IMPORTE_COMPRA, PRECIO_FINAL.....

Real TANTO_POR_CIENTO,.....

.....

fin_entorno

Inicio_Instrucciones

/* En este bloque se representarán las distintas instrucciones que formen parte del bloque de instrucciones del algoritmos que estamos describiendo.*/

Escribir "Indicar el importe de la compra"

Leer IMPORTE_COMPRA

.....

fin_instrucciones

2. Algoritmos.

La representación del entorno de un algoritmo mediante pseudocódigo se realiza de forma análoga a como se hace en los ordinogramas:

Entorno

//Declaración de variables

tipo1 NombreVariable11,...NombreVariable1n;

tipo2 NombreVariable21,...,NombreVariable2n;

...

tipom NombreVariablem1,...,NombreVariablemn;

//Declaración de constantes

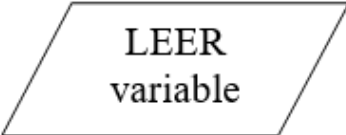
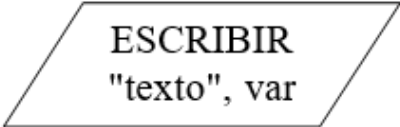
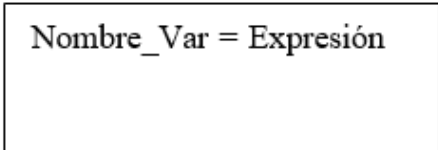
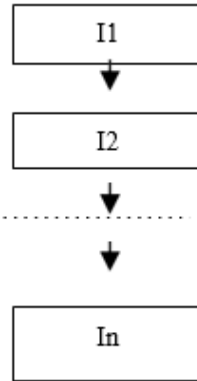
Constante tipo1 NombreConstante11,..., Constante NombreConstante1n;

Constante tipo2 NombreConstante21,..., Constante NombreConstante2n;

...

Constante tipon NombreConstantem1,...,Constante NombreConstantemn;

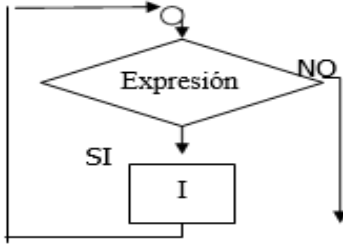
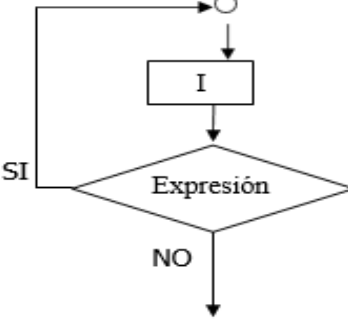
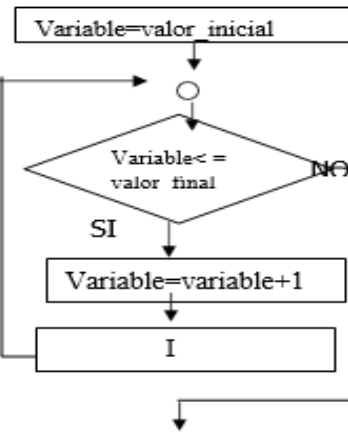
INSTRUCCIONES SECUENCIALES

Categoría de Instrucción	Representación en un ordinograma	Representación en un pseudocódigo
Instrucciones de Entrada		Leer variable;
Instrucciones de Salida		Escribir "texto" Escribir "texto",var,"texto" Escribir "texto",var
Instrucción de asignación		Nombre_Var = Expresión
Bloque secuencial de instrucciones (I1, I2,... In) Siendo Ij instrucciones pertenecientes a las categorías anteriores		I1 I2 In

ALTERNATIVAS

Categoría de Instrucción	Representación en un ordinograma	Representación en un pseudocódigo
Alternativa Simple	<pre> graph TD Entry(()) --> Expresion{Expresión} Expresion -- NO --> Merge(()) Expresion -- SI --> I[I] I --> Merge Merge --> Exit(()) </pre>	<p>Si condición</p> <p>entonces</p> <p style="padding-left: 40px;">I</p> <p style="padding-left: 40px;">.....</p> <p>fin_si</p>
Alternativa Doble	<pre> graph TD Entry(()) --> Expresion{Expresión} Expresion -- NO --> J[J] Expresion -- SI --> I[I] J --> Merge(()) I --> Merge Merge --> Exit(()) </pre>	<p>Si condición</p> <p>entonces</p> <p style="padding-left: 40px;">I</p> <p>si_no</p> <p style="padding-left: 40px;">J.....</p> <p>fin_si</p>
Alternativa Múltiple	<pre> graph TD Entry(()) --> Expresion{Expresión} Expresion --> I1[I1] Expresion --> IN[IN] Expresion --> INplus1[IN+1] </pre>	<p>Según Expresion</p> <p>hacer</p> <p style="padding-left: 40px;">ValorConstante1:I₁</p> <p style="padding-left: 40px;">.....</p> <p style="padding-left: 40px;">ValorConstante2:I₂</p> <p style="padding-left: 40px;">.....</p> <p style="padding-left: 40px;">.....</p> <p style="padding-left: 40px;">ValorConstanten:I_N</p> <p style="padding-left: 40px;">.....</p> <p>fin_segun</p>

REPETITIVAS

Categoría de Instrucción	Representación en un ordinograma	Representación en un pseudocódigo
Instrucción Mientras:	 <pre> graph TD Start(()) --> Expresion{Expresión} Expresion -- SI --> Expresion Expresion -- NO --> Exit(()) </pre>	Mientras Expresion hacer I..... fin_mientras
Instrucción Repetir	 <pre> graph TD Start(()) --> I[I] I --> Expresion{Expresión} Expresion -- SI --> I Expresion -- NO --> Exit(()) </pre>	Repetir I..... mientras Expresion
Instrucción Para	 <pre> graph TD Init[Variable=valor_inicial] --> Expresion{Variable <= valor_final} Expresion -- SI --> Inc[Variable=variable+1] Inc --> I[I] I --> Expresion Expresion -- NO --> Exit(()) </pre>	Para variable=valor_inicial hasta valor_final[ConIncremento ConDecremento] I..... fin_para

3. Metodología de la programación.

3.1 Concepto de programa.

- Un programa es una serie de órdenes o instrucciones ordenadas con una finalidad concreta que realizan una función determinada basándose en un algoritmo predefinido.
- La principal razón por la que las personas aprenden lenguajes y técnicas de programación es la de poder diseñar y realizar programas que resuelvan sus problemas de forma automática una vez que sean ejecutados por un ordenador.
- Tras superar la fase de diseño del programa, el programador se encuentra con la fase de codificación, es decir, hay que pasar del algoritmo diseñado al programa escrito en un determinado lenguaje con su código concreto. Una vez comprobado que el algoritmo es correcto hay que pasar a redactar en un lenguaje de programación (por ejemplo C, C++, Java,..) un programa, que al ser ejecutado por el ordenador realice dicho algoritmo y por tanto resuelva de forma automática el problema planteado.

3. Metodología de la programación.

3.2 Estructura de un programa.

- Antes de conocer las técnicas concretas para describir el programa adecuadamente, es necesario saber cuál es la estructura general de un programa. Como ya hemos visto, la estructura del algoritmo es:

Entorno

.....

fin_entorno

INICIO

.....

FIN

PSEUDOCODIGO:

Entorno

.....

fin_entorno

Inicio

.....

fin

3. Metodología de la programación.

3.2 Estructura de un programa.

- La estructura del programa es básicamente la misma:



3. Metodología de la programación.

3.2 Estructura de un programa.

A la hora de indicar al ordenador lo que tiene que hacer, o lo que es lo mismo, a la hora de describir un programa, éste se compone de dos bloques bien diferenciados:

- Bloque de declaraciones o entorno. En él se especifican o declaran todos los objetos (constantes, variables, tablas, registros, archivos, etc.) que son capaces de almacenar los datos de entrada y/o de salida que utiliza el algoritmo. La declaración de un dato consiste en indicar si es constante o variable, cual va a ser su nombre o identificador y el tipo al que pertenece. En el caso de las constantes habrá que indicar cuál es el valor que tiene asignado.

Ejemplos:

entero X , Edad ;

constante **entero** Numero_total_modulos_primer=3;

real longitud;

constante **real** pi=3.141516;

booleano par;

caracter letra , respuesta;

string calle;

constante **string** Mi_calle= "Pintor Velazquez"

3. Metodología de la programación.

3.2 Estructura de un programa.

- Bloque de instrucciones. Constituido por el conjunto de instrucciones que especifican la secuencia de operaciones a realizar por el procesador en un determinado orden, para que a partir de unos datos de entrada se obtengan unos resultados o datos de salida (las variables y constantes en los que se van a almacenar los datos de entrada y salida deben declararse en el bloque anterior).

El bloque de instrucciones, a su vez está dividido en tres partes fundamentales: Entrada de datos, proceso de datos y salida de resultados. En algunos algoritmos las tres partes del bloque de instrucciones están perfectamente delimitadas, pero en la mayoría de los algoritmos (principalmente los que utilizan proceso interactivo) sus instrucciones quedan entremezcladas a lo largo del algoritmo.

3. Metodología de la programación.

3.2 Estructura de un programa.

a) Entrada de datos.

- Está formada por las instrucciones de entrada.
- Una instrucción de entrada es aquella que tiene como misión tomar, leer o introducir, uno o varios datos desde un dispositivo de entrada (teclado por defecto), almacenarlos en la memoria central en las variables cuyos identificadores aparecen en la propia instrucción.
- Si estas variables tuviesen algún valor previo, éste se perdería. Este tipo de instrucciones se realizará siempre de forma secuencial, es decir, que después de la instrucción de entrada se realizará la siguiente instrucción que esté especificada en el algoritmo.

b) Proceso de datos.

- Está formado por el conjunto de operaciones e instrucciones destinadas a operar con los datos de entrada para poder generar el conjunto de datos o mensajes que forman el resultado de salida.

3. Metodología de la programación.

3.2 Estructura de un programa.

c) Salida de datos.

- Está formada por las instrucciones de salida.
- Una instrucción de salida es aquella que tiene como misión enviar, imprimir, visualizar o escribir datos a un dispositivo de salida (pantalla por defecto), bien tomándolos de variables o constantes almacenados en la memoria principal o bien porque estén definidos de alguna forma en la propia instrucción de salida. Al igual que las instrucciones de entrada las instrucciones de salida se realizan de forma secuencial.
- Ejemplo: Algoritmo para calcular el área de un triángulo, conocidas la base y altura.

$A = b \cdot h / 2$

E/ : b, h

Proc.: $A = b \cdot h / 2$

/S : Resultado según los valores de b y h

3. Metodología de la programación.

3.3 Elementos de un programa.

3.3.1 Constantes y variables.

- En los programas cada dato queda identificado por un nombre o identificador y un valor. A los datos cuyo valor permanece fijo durante la ejecución de un programa se le denomina constantes y a los datos cuyo valor puede variar durante la ejecución de un programa se les denomina variables.
- El nombre de una variable, constante o cualquier objeto definido en el algoritmo (identificador), se formará conforme a las siguientes reglas:
 - Debe resultar significativo, sugiriendo lo que representa.
 - No puede coincidir con palabras reservadas, propias del lenguaje algorítmico.
 - Se admitirá un máximo de caracteres, 32.
 - Comienza siempre con un carácter alfabético, nunca numérico, siguiéndole otros caracteres numéricos o alfabéticos o el símbolo de subrayado (guion bajo) para separarlos.
 - Podrán usarse mayúsculas o minúsculas (cada lenguaje tendrá sus convenios).

3. Metodología de la programación.

3.3 Elementos de un programa.

Ejemplos:

- Nombres o identificadores válidos:

A, x, Edad, Anno, A11, A12, fechaDeNacimiento, Domicilio_actual , Pi

- Nombres o Identificadores no válidos

+, *2, 2izda, 4=3

Es esencial tener muy clara la diferencia entre el nombre de la variable y su contenido.

3. Metodología de la programación.

3.3.2 Tipos de datos: simples y estructurados.

- Trabajar con la representación binaria de los datos es muy difícil por lo que los lenguajes de programación y las técnicas de representación de algoritmos permiten ignorar los detalles de la representación binaria de los mismos y definen los datos como cualquier objeto manipulable por un programa escrito en dicho lenguaje o técnica. Por tanto, los programadores que utilicen un lenguaje de programación o una técnica de representación de algoritmos no verán los datos como cadenas de 0's y 1's sino como un carácter introducido por teclado, una cantidad que representa el radio de un círculo, la altura de una persona, etc.
- Los datos que manipula un algoritmo están organizados en unos conjuntos lógicos denominados tipos de datos. Cada tipo de dato queda definido por los valores que pueden tomar los datos que pertenecen a dicho conjunto, así como por las operaciones que se pueden realizar sobre dichos valores.
- Los tipos de datos se pueden clasificar en:
 - Tipos de datos simples.
 - Tipos de datos compuestos (estructurados).

3. Metodología de la programación.

A) Los tipos de datos simples son:

1. Numéricos. Formados por aquellos datos que toman valores numéricos y sobre los que se pueden realizar operaciones aritméticas como sumas, productos, etc. A su vez se dividen en:
 - Entero: Son un subconjunto dentro de los números reales, que no tienen componentes decimales y pueden ser negativos o positivos, incluido el cero: -2, -1000, 0, 1, 2, 3... Normalmente 1 entero ==> 2bytes ; -32768 al 32767 ($2^{16}=65536$ representaciones)
 - Real: Es el conjunto de números que tienen una parte entera y otra decimal, separadas por un punto (una coma en álgebra no automatizada) y que no posee parte imaginaria. Ej.: 1, 3.1415..., -12.45, ...

Según la representación o la precisión del ordenador, las operaciones serán más o menos exactas, ya que los números reales pueden poseer infinitos decimales, incluso los números enteros, pueden ser muy grandes o muy pequeños y no tener cupo en la forma de representación elegida o disponible.

Ej: $1/3=0.33333333333333...$ 9876^{12345}

3. Metodología de la programación.

2. Tipo lógico o booleano. Cualquier dato perteneciente a este tipo sólo puede tomar uno de los dos valores siguientes: verdadero (V) o falso (F).
 - Ej.: SOLTERO $\Rightarrow \{V, F\} \equiv \{1, 0\}$
3. Caracter. Un dato del tipo caracter contiene uno de los caracteres del juego de caracteres ASCII, que se clasifican en tres grupos:
 - Caracteres alfabéticos ('A'...'Z'; 'a'...'z') dígitos numéricos ('0'...'9')
 - Caracteres especiales: ('+' '-' '*' '.' ';' '<' '>' ,...)
- Ej.: VOCAL = 'a'

3. Metodología de la programación.

4. Cadena de caracteres o cadenas alfanuméricas (string). Una cadena de caracteres es una sucesión de caracteres que se encuentran delimitados por unas comillas dobles, para diferenciarlas de los nombres de las variables y constantes.
 - Ej.: "HOLA MUNDO", "22 de octubre de 2001".
 - La longitud de una cadena de caracteres viene determinada por el número de caracteres que hay comprendidos entre los dos símbolos de comillas incluidos los espacios en blanco.
 - Ej.: "Hola, que tal" -> longitud del string 13.
"Este es el valor de VOL_CUBO" -> longitud del string 28.
"12345.73" -> longitud del string 8.

B) Los tipos de datos compuestos o estructurados los iremos viendo a medida que vayamos avanzando. Se trata de datos complejos que se forman por composición de otros datos simples. Podemos citar inicialmente: conjuntos, arrays (vectores y tablas), registros, listas, pilas, colas, etc.

3. Metodología de la programación.

3.3.3. Operadores, funciones internas y expresiones.

• Operadores.

- La manipulación de los datos de entrada que realiza un algoritmo se hace aplicando sobre los mismos una o más operaciones.
- Las operaciones que se pueden realizar con los datos (variables o constantes) de un determinado tipo se representan mediante los operadores . Los principales operadores son:

a) Operadores aritméticos:

- \wedge ó $**$ (potencia) Ej.: $3^4=81$
- $*$ (producto) Ej.: $67*2=134$
- $/$ (división) Ej.: $6/3=2$
- div (división entera) Ej.: $5\text{div}3=1$ ($Z/Z \Rightarrow \text{div}$)
- mod (resto de una división) Ej.: $5\text{mod}3=2$ (En Java : %)
- $+$ (suma) Ej.: $12+4=16$
- $-$ (resta). Ej.: $1-2= -1$

3. Metodología de la programación.

b) Operadores alfanuméricos:

- & ó + Operador de concatenación (unión):

Ejemplo: String C, D, E ;

- C = "Hola y " D = "Adiós"
- E = C & D (El valor de E = "Hola y Adiós")

c) Operadores lógicos:

- not no, negación (En Java es !)
- and y lógico o intersección (En Java es & &)
- or o lógico o unión (En Java es | ||)

3. Metodología de la programación.

A	! A
V	F
F	V

Siendo A una expresión booleana.

A	B	A and B
F	F	F
F	V	F
V	F	F
V	V	V

Siendo A y B expresiones booleanas.

A	B	A or B
F	F	F
F	V	V
V	F	V
V	V	V

Siendo A y B expresiones booleanas.

3. Metodología de la programación.

d) Operadores relacionales:

>	mayor
>=	mayor o igual
<	menor
<=	menor o igual
=	igual
<>	distinto

Ej.: $X=7$

(En Java es **==**)

Ej.: $6 \bmod 3 <> 0$

(En Java es **!=**)

3. Metodología de la programación.

- Los operadores relacionales permiten realizar comparaciones de valores asociados a los tipos numéricos, de tipo carácter, de tipo string y de tipo booleano. El resultado que se obtiene al aplicar un operador de relación, es verdadero o falso.
- En el caso de valores numéricos, el orden que se tiene en cuenta a la hora de evaluar un operador relacional es el de los propios valores, así:
- $3 < -2$ es falso, $5.6 < 7.9$ es verdadero, $4 = 9$ es falso, $4.67 = 4.67$ es verdadero, etc. Para los valores numéricos, es interesante recordar que los ordenadores para almacenar un número real, utiliza un número determinado de posiciones de memoria, así, existen algunos números que no pueden ser almacenados con su valor exacto y deben ser truncados ($1/3 = 0.333\dots$), en consecuencia, las expresiones lógicas de igualdad pueden dar un valor falso aún cuando sean algebraicamente iguales, como por ejemplo:
 - $(1 / 3) * 3 = 1$ es una expresión falsa.

3. Metodología de la programación.

- En el caso de los caracteres, el orden que se tiene en cuenta a la hora de evaluar un operador relacional es el que proporciona el propio código ASCII, en el que:

1º.) Los números del cero al nueve tienen su orden natural. '0' < '1' < '2' ... < '8' < '9'

2º.) Las letras mayúsculas de la 'A' a la 'Z' en orden alfabético. '9' < 'A' < 'B' < 'C' ... < 'Y' < 'Z'

3º.) Las letras minúsculas de la 'a' a la 'z' en orden descendente. 'Z' < 'a' < 'b' < 'c' ... < 'y' < 'z'

'1' < '2' es verdadero, 'A' < 'B' es verdadero, 'a' < 'g' es verdadero, etc.

- En el caso de valores booleanos, el orden que se tiene en cuenta a la hora de evaluar un operador relacional es considerar que falso < verdadero así si tenemos:

Boolean A, B

A=verdadero B=falso

A < B es falso

B < A es verdadero

3. Metodología de la programación.

- **Funciones internas.**
- Además de utilizar los operadores para realizar operaciones sobre los datos de un algoritmo, en numerosas ocasiones, es necesario utilizar unos operadores especiales, llamados funciones internas, y que se pueden utilizar sin necesidad de definirlos. Entre estas funciones internas predefinidas están por ejemplo:
 - $\text{abs}(x) \Rightarrow$ valor absoluto
 - $\text{sen}(x) \Rightarrow$ seno
 - $\text{cos}(x) \Rightarrow$ coseno
 - $\text{raiz2}(x) \Rightarrow$ raíz cuadrada
 - $\text{exp}(x) \Rightarrow$ e elevado a x
 - $\text{ent}(x) \Rightarrow$ parte entera
 - $\text{ln}(x) \Rightarrow$ logaritmo neperiano

3. Metodología de la programación.

- **Expresiones.**

- Una expresión es la representación de un cálculo necesario para la obtención de un resultado. Una expresión se puede definir de alguna de las siguientes maneras:

- i) Un valor es una expresión:

- Ej.: 12, 1.23, "Antonio", 'd', verdadero

- ii) Una variable o una constante es una expresión:

- Ej.: Dia_Semana, A, pi

- iii) Una función interna es una expresión:

- Ej.: sen(pi)

- iv) Cualquier combinación válida de operadores, constantes, variables y/o funciones internas, así por ejemplo serán expresiones las siguientes:

- Ej.: cos(pi*x)+9 'f'<'d' "Juan y Pedro"

3. Metodología de la programación.

- Cada expresión toma un valor que se determina evaluando cada una de las constantes, variables y funciones internas implicadas y realizando las operaciones indicadas, en la misma.
- Teniendo en cuenta el tipo al que pertenece el resultado obtenido tras evaluar una expresión, éstas se pueden clasificar en:
 - Numéricas:
 - Son las que producen un resultado numérico y se construyen mediante los operadores aritméticos. Ej.: $(3+4)^{3/5}$
 - Alfanuméricas:
 - Son las que producen un resultado de tipo alfanumérico y se construyen mediante el operador alfanumérico. Ej.: "Blanco" & " y " & "Negro"
 - Booleanas:
 - Son las que producen un resultado booleano (verdadero o falso) y se construyen mediante los operadores relacionales y lógicos. Ej.: $(3=6/2)$ and $(respuesta='s')$

3. Metodología de la programación.

- Como en una expresión puede aparecer más de un operador, para calcular el valor de la misma es necesario establecer unas reglas de prioridad que permitan decidir qué operador se aplica primero. Estas reglas son las siguientes:
 1. Las expresiones encerradas entre paréntesis se evalúan primero. Si existen varios paréntesis anidados los internos se evalúan primero.
 2. El operador - unitario (que cambia de signo)
 3. Potencias.
 4. Productos y divisiones.
 5. Sumas y restas.
 6. Concatenación.
 7. Relacionales.
 8. Negación. not
 9. Conjunción. (operador &&) and
 - 10.- Disyunción. (operador ||) or
- La evaluación de operadores de igual prioridad en orden siempre se realiza de izquierda a derecha.

3. Metodología de la programación.

- Un paréntesis, siempre nos sirve para dar prioridad a una operación. Por tanto siempre que dudemos, es conveniente usar paréntesis, ya que éstos siempre servirán para asegurarnos ante la duda, de que se realiza una operación antes que otra, según nuestras necesidades.

- Ejemplo:

$$((3+2)^2-15)/2*5=$$

$$(5^2-15)/2*5 =$$

$$(25-15)/2*5 =$$

$$10/2*5 =$$

$$5*5 = 25$$

- Evaluar: a.) $3+6*14$, b.) $(-4*7)+2^3/4-5$, c.) $8/8*2$

3. Metodología de la programación.

• Solución:

a.) $3+6*14$

1.) $6*14=84$

2.) $3+84=87$

b.) $(-4*7)+2^3/4-5$

1.) $-4*7= -28$

2.) $2^3=8$

3.) $8/4=2$

4.) $-28+2= -26$

5.) $-26-5= -31$

c.) $8/8*2$

1.) $8/8=1$

2.) $1*2=2$

no hacer:

1.) $8*2=16$

2.) $1/16=0'0625$

3. Metodología de la programación.

3.3.4. Instrucciones.

- **Instrucciones de asignación.**

- La instrucción de asignación es aquella que nos permite darle valores a una variable y se representa con el símbolo "=". Al igual que las instrucciones de entrada y de salida, las instrucciones de asignación se realizan de forma secuencial. El formato general es:

nombre de variable = expresión o también

A = 5 (el valor de la variable A es 5)

- La acción de asignar es destructiva, ya que el valor que tuviera la variable antes de la asignación se pierde y se reemplaza por el nuevo valor:

1. A = 5

2. A = 7

3. A = 33

- El último valor que tomará A es 33, ya que los valores 5 y 7 se perderán.
- En caso de asignar a una variable una expresión compleja, se evalúa en primer lugar la expresión y ese valor es el asignado a la variable en cuestión: $A = 8/4 + 1 \dots$ A vale 3.
- Es posible utilizar el mismo nombre de variable a ambos lados de la asignación, así, acciones como: $N = N + 1$, tienen sentido, y el valor que tomaría N después de la asignación es una unidad mayor al que tenía antes de la asignación. (No se trata de una ecuación matemática: N nunca podría ser igual a $N + 1$, sino de una asignación N vale lo que valía antes más uno).

3. Metodología de la programación.

- Ejemplo:

1. $A = 12$

2. $A = A/2$ (A toma el valor 6)

- Antes de poder utilizar una variable es necesario inicializarla previamente con un valor, la forma más habitual de realizar esta inicialización es mediante una instrucción de asignación (otra forma es mediante la instrucción de entrada LEER variable).
- Una asignación puede ser aritmética ($A = 3$), lógica ($PAR = F$), caracter ($vocal = 'e'$) o cadena ($nombre = "Pedro"$).
- A una variable no se le pueden asignar valores de distinto tipo del que fue definida dicha variable, ya que en ese caso estamos cometiendo un error de tipo.

Ej.: entero x, y, z

$z = x / y$

//No siempre el cociente de dos enteros es entero

3. Metodología de la programación.

- **Instrucciones de control.**
- Son instrucciones que no realizan trabajo u operaciones efectivas salvo la evaluación de expresiones, generalmente lógicas, con el objetivo de controlar la ejecución de otras instrucciones o alterar el orden de ejecución normal de las instrucciones de un algoritmo.
- Existen dos grandes grupos de instrucciones de control:
- Instrucciones alternativas o decisiones.
- Son aquellas que controlan la realización de uno o varios bloques de instrucciones, dependiendo del cumplimiento o no de alguna condición o del valor final de una expresión. Una decisión permite por tanto romper la secuencia del algoritmo.

3. Metodología de la programación.

- Existen tres modelos típicos de instrucciones alternativas:
 - Alternativa simple: si se cumple una condición (/es) se realiza una acción (/es).

Ej.: **Si** llueve

entonces

cogeré un paraguas

fin_si;

3. Metodología de la programación.

- Alternativa doble: si se cumple una condición(es) se realiza una acción(/es), si no se realiza otra acción(/es).

Ej.: **Si** llueve

entonces

cogeré un paraguas

sino

esperaré a que no llueva

fin_si;

- Alternativa múltiple: según el valor de una condición o expresión se realizará una acción dentro de las múltiples posibles, asociadas al valor de la condición.

Ej.: **Segun** dia **hacer** I1, I2,...instrucciones.

lunes: I1

martes: I2

.....

fin_segun;

- Ejemplos:
- 1. Dado un número por teclado, decidir si es par o impar.

Algoritmo PAR_IMPAR

//Determina si un número introducido es par

Entorno

Entero A

fin_entorno

Inicio

Escribir "Introduce un número:"

Leer A

Si ($A \bmod 2 = 0$) entonces

Escribir "el número", A, "es par"

sino

Escribir A, "es un número impar"

fin_si

fin

2. Dados dos números por teclado, decir si un número es múltiplo de otro:

Algoritmo MULTIPLO

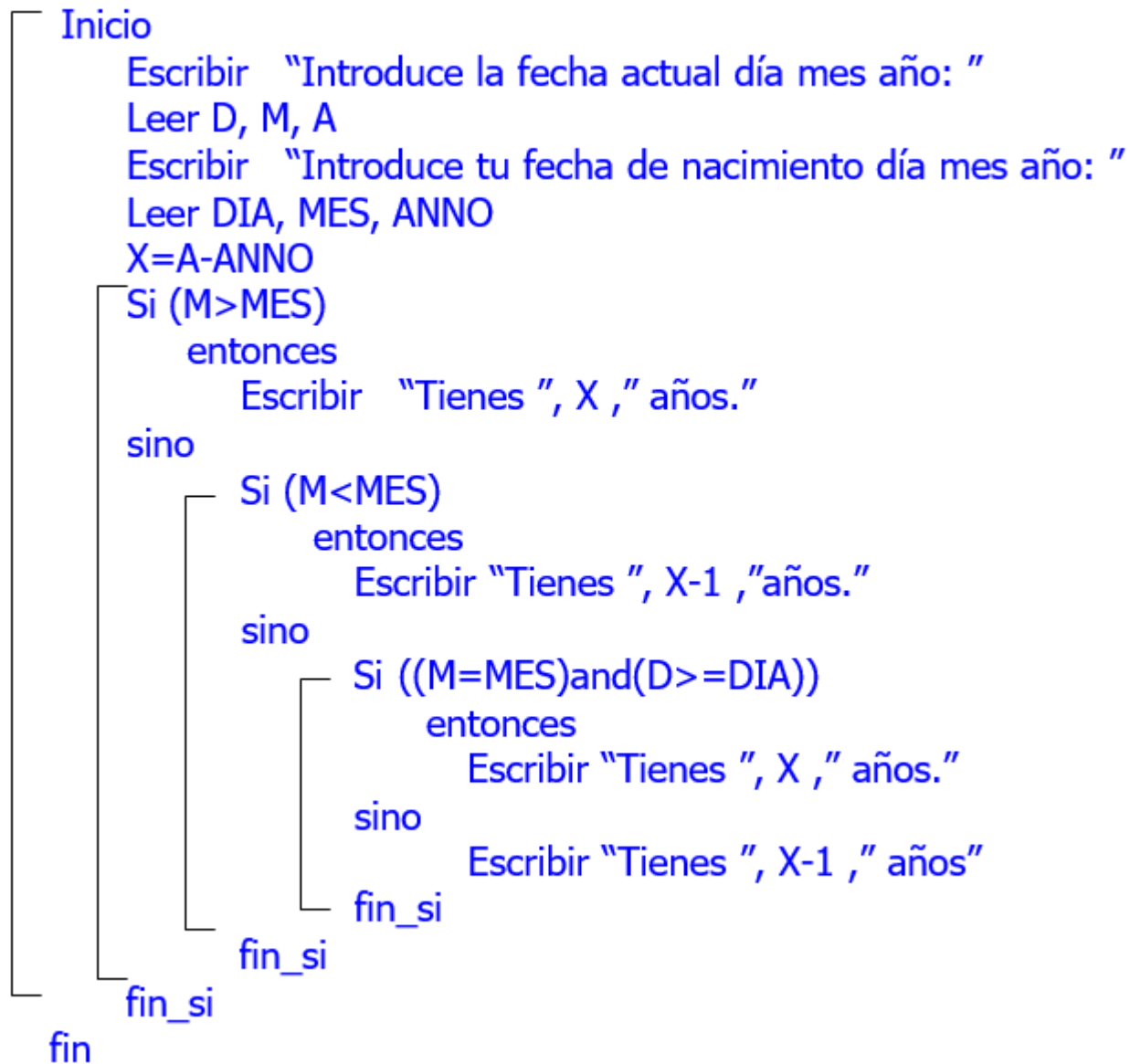
```
— //Determina si un número es múltiplo de otro
Entorno
    Entero A,B
fin_entorno

Inicio
    Escribir "Introduzca dos números:"
    Leer  A,B
    Si ((A>B)and((AmodB)=0))
        entonces
            Escribir B,"es múltiplo de",A
        sino
            Si ((B>A)and((BmodA)=0))
                entonces
                    Escribir A,"es múltiplo de",B
                sino
                    Si (A=B)
                        entonces
                            Escribir "son iguales"
                        sino
                            Escribir "no son múltiplos"
                    fin_si
                fin_si
            fin_si
        fin_si
    fin
fin
```

3. Dado el día, mes y año de nacimiento de una persona y dado también el d, m, a actual, nos diga su edad:

Algoritmo EDAD

```
— //Determina la edad de una persona conocida la fecha de nacimiento
Entorno
    Entero D, M, A, DIA, MES, ANNO /*datos actuales: D, M, A y
                                   de nacimiento: DIA, MES, ANNO */
    Entero X // Edad
fin_entorno
```



Este programa podría realizarse de forma más simple agrupando condiciones:

```
Si ((M>MES) or ((M=MES)and(D>=DIA)))  
    entonces  
        Escribir "Tienes ", X ," años."  
sino  
    Escribir "Tienes ", X-1 ,"años."  
fin_si
```


4. Realiza la suma, resta, cociente o división de dos números:

Algoritmo CALCULADORA

//Calcula el resultado de la operación aritmética elegida sobre dos operandos
elegidos.

— Entorno

Real X,Y

Caracter OPCION

Booleana SALIR

fin_entorno

4. Realiza la suma, resta, cociente o división de dos números:

```
Inicio
  Escribir "Introduce el primer operando: "
  Leer X
  Escribir "Introduce el segundo operando: "
  Leer Y
  SALIR='f'
  Repetir
    Mientras (OPCION>='a' or OPCION<'f')
      Escribir " Elija opción a, b, c, d, e: "
      Escribir "      a) SUMA. "
      Escribir "      b) RESTA. "
      Escribir "      c) COCIENTE. "
      Escribir "      d) DIVISIÓN. "
      Escribir "      e) SALIR. "
      Leer OPCION
    fin_mientras

    Segun OPCION hacer
      a: Escribir " La suma ", X, " + ", Y, " = ", X + Y
      b: Escribir " La resta ", X, " - ", Y, " = ", X - Y
      c: Escribir " El cociente ", X, " / ", Y, " = ", X / Y
      d: Escribir " El producto ", X, " * ", Y, " = ", X * Y
      e: SALIR='v'
    fin_segun
  hasta SALIR='v'
fin
```

- Instrucciones repetitivas.
 - Son aquellas que controlan la repetición de un conjunto o bloque de instrucciones mediante la evaluación de una condición que se realiza cada nueva repetición, por medio de un contador asociado.
 - Son instrucciones que repiten una secuencia de instrucciones un número determinado de veces, también se las llama bucles.
 - Existen básicamente tres tipos de instrucciones repetitivas o bucles que son: la instrucción mientras, la instrucción repetir y instrucción para:
- Instrucción mientras. La instrucción repetitiva mientras es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición, es decir, realiza una iteración cada vez que la condición resulta ser cierta. Las palabras clave que indican una instrucción mientras en pseudocódigo son:

Mientras CONDICION

ACCION 1

ACCION 2

ACCION 3

fin_mientras

Ejemplo:

Factorial de un n°

Algoritmo FACTORIAL

```
// Algoritmo que calcula el factorial de un número natural
Entorno
  Entero NUM, INI, FAC
fin_entorno
Inicio
  FAC = 1
  Escribir "Introduce un número mayor que 1"
  Leer NUM
  INI=NUM
  Mientras NUM > 1
  [
    FAC = FAC * NUM
    NUM = NUM - 1
  ]
  fin_mientras
  Escribir "El factorial de", INI , " es ", FAC.
Fin
```

Lo primero que se hace al entrar en una instrucción mientras es la evaluación de la condición y si resulta ser falsa el cuerpo del bucle nunca se ejecutará. Si la condición es cierta se ejecuta una iteración y después se vuelve a evaluar la condición.

- Instrucción repetir. La instrucción repetitiva repetir es aquella en que el cuerpo del bucle se ejecuta mientras que se cumple una determinada condición, es decir, realiza una iteración cada vez que la condición resulta ser verdadera. Las palabras clave que indican una instrucción repetir en pseudocódigo son:

Repetir

ACCIÓN 1

ACCIÓN 2

...

mientras CONDICION

Ejemplo:
Factorial de un n°

```
Algoritmo FACTORIAL
    // Algoritmo que calcula el factorial de un número natural
    Entorno
        Entero NUM, FAC
    Fin_entorno
    Inicio
        FAC = 1
        Escribir "Introduce un número mayor que 1"
        Leer NUM
        Repetir
            FAC = FAC * NUM
            NUM = NUM - 1
            mientras NUM > 1
        visualizar NUM
    fin
```

La principal diferencia con la instrucción mientras es que lo primero que se ejecuta en la instrucción, es el bucle y después se evalúa la condición, si resulta ser verdadera se ejecutará de nuevo, volviendo a evaluar al final la condición hasta que resulte ser falsa. En la instrucción repetir el bucle se ejecutará una vez al menos, es decir, realiza como mínimo una iteración.

- Instrucción para: En muchas ocasiones se conoce de antemano el número de veces que se desea ejecutar las instrucciones del bucle. En estos casos en que el número de iteraciones es fijo y se debe usar la instrucción para. La instrucción para se ejecutará un número determinado de veces y de modo automático controlará el número de iteraciones del cuerpo del bucle. Sus palabras clave en pseudocódigo son:

```
Para VARIABLE = VALOR-INI. hasta VALOR-FIN. [ incr. INC. | decr. DEC.]  
    ACCION 1  
    ACCION 2  
    .....  
fin_para
```

La instrucción para comienza con un valor inicial, las acciones específicas se ejecutan a menos que el valor inicial sea mayor (o menor si se decrementa) que el valor final. La variable se incrementa en INC. o decrementa en DEC., si este nuevo valor no supera el final se ejecutarán de nuevo las acciones. El incremento de la variable será siempre de 1 a no ser que se especifique lo contrario.

Ejemplo:

Factorial de un n^o (sólo especificamos el bloque de instrucciones)

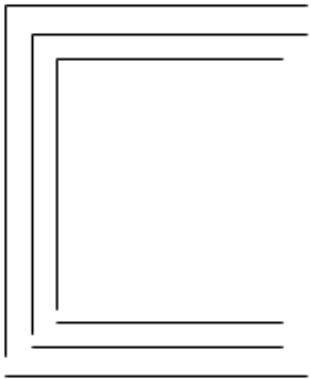
```
Inicio
    Escribir " Dame un n°: "
    Leer NUM
    FAC = NUM
    Para I = NUM-1 hasta 2 decremento 1
        FAC = FAC * I
    fin_para
    Escribir "El factorial de ", NUM, " es : ",FAC
fin
```

Podemos realizar el mismo algoritmo con incremento:

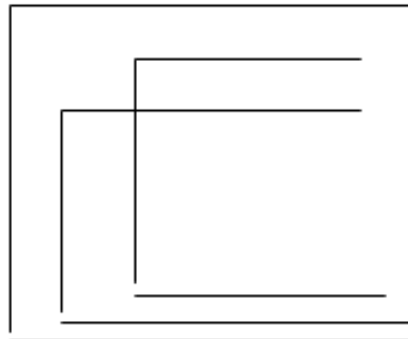
```
Inicio
    Escribir " Dame un n°: "
    Leer NUM
    FAC = 1
    Para I = 2 hasta NUM incremento 1
        FAC = FAC * I
    fin_para
    Escribir "El factorial de ", NUM, " es : ",FAC
fin
```


- Las instrucciones alternativas y/o repetitivas pueden anidarse unas con otras tantas veces como sea necesario, aunque, por claridad en el pseudocódigo, no es recomendable anidar más de tres instrucciones repetitivas, en caso de necesitar más anidaciones dividiremos el problema inicial en módulos para facilitar la labor de programación.
- Las anidaciones deben hacerse de forma correcta, esto es, el inicio y fin del bucle más interno no deben superar los del más externo.

Bien



Mal



Ejemplo:

Sumar los números impares menores que 1000.

Algoritmo SUMIMPAR

//Suma los impares menores de 1000

Entorno

Entero I, SUMA

fin_entorno

Inicio

SUMA = 0

Para I = 1 hasta 999 incremento 2

SUMA = SUMA+I

fin_para

Escribir "La suma de los impares menores de 1000 es : ", SUMA

fin

- **Subprogramas.**

Una subrutina es un fragmento del programa que realiza una tarea concreta y recibe un nombre por el que puede ser llamada o activada desde otra parte del algoritmo. Una subrutina puede tener una serie de variables de comunicación denominadas argumentos, que permiten el paso de información entre el algoritmo y la subrutina.

El uso de subrutinas:

- Evita la duplicación de grupos de instrucciones en diferentes partes del algoritmo
- Facilita la construcción y comprensión de los algoritmos, ya que dividimos el algoritmo en varias partes de forma que:
- Cada una de ellas es más fácil de describir que el algoritmo completo
- Cada una de ellas de forma individual realiza una función concreta y todas juntas resuelven el problema planteado inicialmente.

Podemos indicar que existen dos tipos de subrutinas: los procedimientos y las funciones. Éstas se diferencian esencialmente de aquellos, en que retornan algún resultado o valor de salida.

3. Metodología de la programación.

3.4 Variables auxiliares.

- Son objetos que utiliza un algoritmo y que por la frecuencia con la que se utilizan y por la función que realizan dentro del mismo, toman un nombre especial: contadores, acumuladores e interruptores o switches.

3.4.1. Contadores.

- Un contador es una variable destinada a almacenar un valor que se irá incrementando o decrementando en una cantidad constante.
- Es una variable que se utiliza para contar cualquier evento que pueda ocurrir dentro del algoritmo o programa.
- Se suelen utilizar mucho en procesos repetitivos, es decir en los cuerpos de las instrucciones repetitivas.
- Sobre un contador se realizan dos operaciones básicas:

Inicialización:

Todo contador se debe inicializar con un valor inicial (0, 1...)

Ej.: `CONTADOR = Valor_Inicial`

Incremento

Cada vez que aparezca el evento a contar se ha de incrementar o decrementar en una cantidad fija (I, D respectivamente) el valor del contador.

`CONTADOR = CONTADOR + I` `CONTADOR = CONTADOR - D`

Ejemplo:

Cuenta el nº de A's que un usuario introduce hasta finalizar con la pulsación de un *

Algoritmo VOCAL_A

// Este algoritmo calcula el número de A's introducidas por el usuario

```
Entorno
  Entero CONT
  Caracter LETRA
fin_entorno
Inicio
  CONT = 0
  Mientras (LETRA <> '*') hacer
    Escribir "Introduce una A o un *"
    Leer LETRA
    Si LETRA = 'A' entonces
      CONT = CONT + 1
    fin_si
  fin_mientras
  Escribir CONT
fin
```

Observaciones:

Cuando se utiliza un contador con una instrucción Repetir o Mientras la operación de inicialización se realiza fuera del cuerpo del bucle y la de incrementación o disminución se realiza dentro. Las variables de control del bucle Para son un ejemplo de variable contador en las que la inicialización y el incremento se realizan de forma automática.

3.4.2. Acumuladores.

- Un acumulador o totalizador es una variable cuya misión es acumular cantidades sucesivas obtenidas al realizar la misma operación. Realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante como en el caso del contador. El uso más habitual de un acumulador es obtener sumas y productos.
- Al igual que con los contadores, para poder utilizar un acumulador hay que realizar sobre ellos dos operaciones básicas:

Inicialización:

Todo acumulador necesita ser inicializado con el valor neutro de la operación que va a acumular, que en el caso de la suma es el 0 y en el caso del producto es el 1.

SumaTotal=0; ProductoFinal=1;

Acumulación:

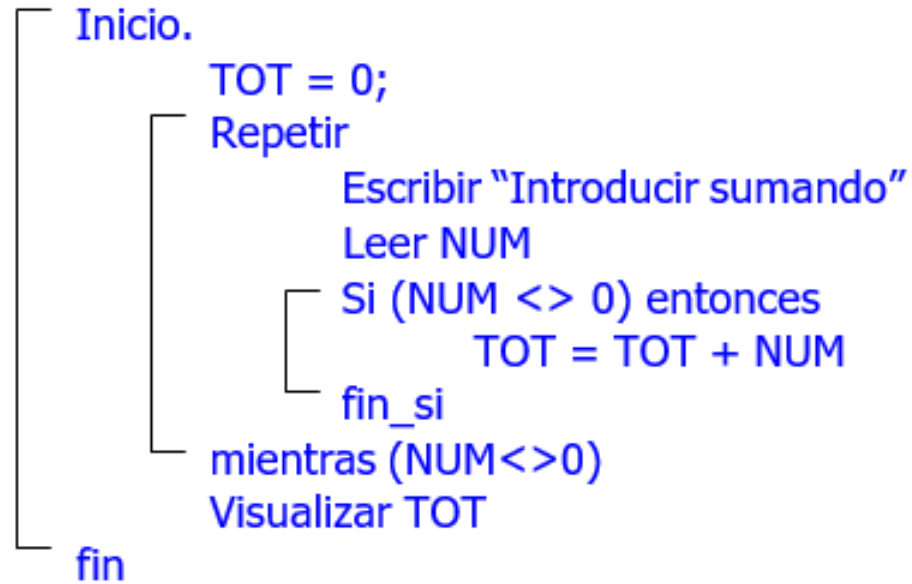
Una vez obtenido y almacenado en una variable la cantidad a acumular, le añadimos a la variable acumulador dicha cantidad, y le reasignamos el resultado.

SumaTotal = SumaTotal + cantidad;

ProductoFinal = ProductoFinal + cantidad;

Ejemplo:

Suma una serie de números introducidos por teclado hasta que se introduce un 0.



3.4.3. Switches.

- Un interruptor, conmutador o switch es una variable que puede tomar dos posibles valores a lo largo de la ejecución del programa. Los valores que toma son normalmente 1 o sí (encendido/abierto) y 0 o no (apagado/cerrado) (de ahí su nombre de interruptor).
- Se utilizan principalmente para:
 - a) Recordar en un determinado lugar del programa la ocurrencia o no de un suceso:

Ejemplo:

Algoritmo que lee una secuencia de notas (hasta que se introduzca el valor -1) y nos dice si hubo o no una nota con valor diez:

```
Algoritmo CONMUTADOR1
Entorno
    Real NOTA
    Booleano ES_SOBRESALIENTE
fin_entorno
Inicio
    ES_SOBRESALIENTE = falso
    Repetir
        Escribir " Introduce una nota"
        Leer NOTA
        Si (NOTA =10)
            ES_SOBRESALIENTE = verdadero
        fin_si
    mientras (NOTA
        != -1)
        Si (ES_SOBRESALIENTE = verdadero) entonces
            Escribir " Al menos hay una nota que es un 10"
        sino
            Escribir " Ninguna nota ha sido un10"
        fin_si
    fin
```


b) Realizar de forma alternativa e independiente dos procesos alternativos

Ejemplo:

Sumar por un lado los números pares comprendidos entre 1 y 100 y por otro los impares:

```
Algoritmo CONMUTADOR2
Entorno
    Entero SUMAPARES, SUMAIMPARES, I
    Booleano ES_PAR
fin_entorno
```

```
Inicio
    SUMAPARES = 0
    SUMAIMPARES = 0
    ES_PAR = falso
    Para I = 1 hasta 100 incremento 1
        Si (ES_PAR = falso ) entonces
            SUMAIMPARES=SUMAIMPARES + I
            ES_PAR = verdadero
        sino
            SUMAPARES = SUMAPARES + I
            ES_PAR = falso
        fin_si
    fin_para
    Escribir " La suma de los primeros 100 impares es:", SUMAIMPARES
    Escribir " Y la de los primeros 100 pares es:",SUMAPARES
fin
```