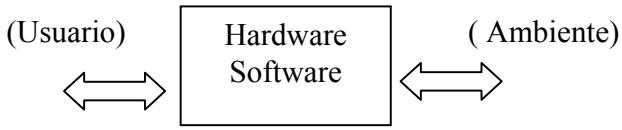
Unidad 1
INTRODUCCIÓN: ALGORITMOS Y ESTRUCTURAS DE DATOS
Desarrollo de la unidad : 12 h

Prácticas : Edición, compilación, Enlace y pruebas.
Ver código máquina, ensamblador y C
Ejercicios : Paso de Ensamblador a código máquina y viceversa. (Didag 94.0)
Paso de alto nivel a ensamblador y viceversa
Elegir distintos tipos de datos en función del problema
Evaluar expresiones y operaciones con datos
Conceptos:
Sistema informático, Fases en el desarrollo de software, Elaboración de programas, Estructura de un programa, Lenguajes de programación, Algoritmo y Estructuras de datos.

Introducción Sistema informático

Informática: Conjunto de conocimientos científicos y técnicos que hacen posible el tratamiento automático de la información por medio de ordenadores.

Ordenador: Máquina polivalente capaz de realizar un conjunto de operaciones aritméticas y lógicas definidas en un programa leyendo unos datos de entrada y generando unos resultados o datos de salida de forma automática.

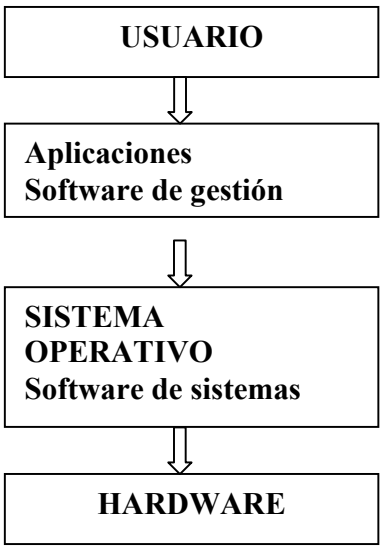


Software : Conjunto de programas, datos y aplicaciones que indican las operaciones que debe realizar un ordenador

¿Donde está el software? -> En la Memoria Principal y en el Memoria Secundaria
El software es inmaterial es un elemento lógico, que tiene que estar almacenado en un soporte físico para que el hardware acceda a él. El hardware sin software es inútil

- Software de sistemas o básico: Sistemas Operativos, controladores de dispositivos, comunicaciones, relacionado con el uso y control del hardware del sistema.

- Software de gestión o aplicaciones: Trata de resolver problemas concretos de usuarios, normalmente relacionados con la gestión administrativa.



Tipos de problemas resuelto por los ordenadores:

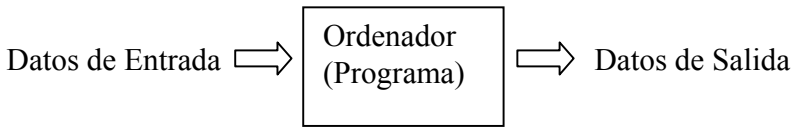
- Rutinarios
- Calculo matemático intensivo
- Tratamiento sencillo de múltiples datos
- Problemas donde no hay incertidumbre, incoherencias, donde existen métodos claros y precisos de resolución.

Ventajas

- Más rápidos,
- No se cansan,
- Menos errores, no tienen errores por distracción o cansancio
- Ahorran tiempo, dinero, (Personal y espacio)

Los sistemas informáticos son Imprescindible y estratégicos en grandes sectores económicos, sobretodo en aquellos que trabajan con información (financiero, distribución, bancario, administración de grandes empresas)

Esquema de funcionamiento de Sistema Informático (S.I.)



Programa = Conjunto de instrucciones, método o algoritmo que aplicado sobre unos datos realizan una determinada operación, resuelven un problema o ofrecen unos resultados.

Desarrollo de Software.

Evolución:

- Primeras décadas (50,60) Principal costo era el hardware
  Existen más programadores que ordenadores

- Presente: principal costo es el software
  Existen más ordenadores que programadores

Comparación entre la producción de hardware y software:

- El hardware se diseña y se fabrica, el software se diseña y se distribuye.
- El hardware sufre un desgaste y un deterioro con el uso, al ser un sistema físico. El software es lógico no se desgasta (aunque puede quedarse desactualizado, anticuado)
- El hardware se realiza uniendo elementos probados que funcionan por separado (procesador, circuitos, chips, placas), El software no es tan sencillo, muchas veces hay que diseñarlo a medida, en función de cada problema.
- El principal coste del software es su diseño, la producción (hacer copias) y distribución (hoy suele realizarse mediante la red Internet) son mucho menores.

Crisis de software

Desde la aparición de los ordenadores el desarrollo de software ha tenido un crecimiento exponencial en número de programas, en tamaño y complejidad de los mismos.

En un principio el desarrollo de software era una labor casi artesanal, cada programador diseñaba su programa según sus gustos, experiencia o costumbre. Existían pocas aplicaciones, muchas veces de mala calidad, que generaban enormes gastos de mantenimiento, siendo la productividad por programador muy baja. El desarrollo de una nueva aplicación era difícil de planificar (evaluar tiempo, recursos humanos, costes, rendimientos finales), casi siempre se obtenía unos resultados deficientes, con grandes problemas en el mantenimiento posterior.

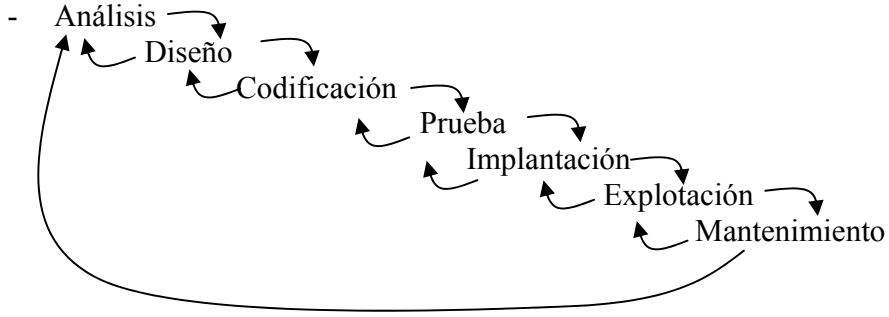
Solución: ***INGENIERÍA DE SOFTWARE***
Definir una serie de métodos, herramientas y procedimientos para desarrollar software que permitan la planificación, la evaluación de costos, y que garanticen la fiabilidad del producto.

Dificultades de los proyectos informáticos

- Tamaño del proyecto
- Rápida evolución del sector
- Complejidad de los programas y entornos de desarrollo
- Problemas de coordinación de equipo de trabajo de desarrollo de software.
- Problemas de comunicación entre en quien encarga el programa y quien lo realiza
  (Cliente – equipo de desarrollo)

Fases del desarrollo o ciclo vida de una aplicación informática

La teoría de Ingeniería de Software ha definido una serie de fases que se deben realizar para desarrollar una aplicación informática. En cada fase se utilizan distintas metodologías y herramientas de trabajo que producen documentos con los que iniciar la siguiente fase.



Análisis

Se define todos los requerimientos que deben cumplir la aplicación, cuales deben ser las entradas y las salidas y que funciones debe realizar. Con que información se va a trabajar, que problemas se van a resolver. En el análisis nos centramos en el QUE , que debe hacer nuestra aplicación informática. Se realiza a partir o conjuntamente con la información que nos ofrece el cliente.

Herramientas : DFD, Definición de pantallas, formatos de informes, Entidades y relaciones

Diseño

Se define las distintas estructuras de datos que almacenan la información: Tablas, ficheros, bases de datos, etc, los distintos algoritmos que trabajan con estos para cumplir los requerimientos que se define en el análisis. En el diseño nos centramos en el COMO, como realizar las operaciones para obtener los resultados descritos en el análisis.

Herramientas: Diagrama de módulos, Pseudocódigos, Diseño de Bases de Datos

Codificación

Consiste en traducir y/o adaptar el diseño de la aplicación a un lenguaje de programación. Si el diseño está detallado la codificación se realiza mecánicamente. En común que en un proyecto se utilicen varios lenguajes de programación.

Pruebas

Una vez elaborado la aplicación informática, se debe realizar un conjunto de pruebas que garanticen que los programas realizan sus funciones correctamente, asegurándose que se obtienen las salidas esperadas a partir de las entradas definidas en el análisis. En la fase de pruebas se determina la calidad de software elaborado.

Tipos:

- Pruebas unitarias
- Pruebas de integración
- Pruebas de sistemas

- Pruebas de integración / o de campo

Implantación

Instalación y configuración de la aplicación en el entorno real para el que ha sido diseñada.

Tipos:

- Directa
- Incremental
- Paralelo
- Implantación piloto

Explotación

Administración y gestión de la aplicación dentro de la empresa. Muchas aplicaciones se deben configurar y gestionar independientemente de la instalación inicial, por ejemplo para dar de alta usuarios, cambiar informes, activar o desactivar funciones, etc.

Mantenimiento

Implica la modificación de la aplicación a lo largo del tiempo (Incluye de nuevo todas las otras fases anteriores)

Tipos:

- Correctivo: Corrección de errores no detectados
- Adaptativo: Cambio de entorno físico o lógico (Sistema operativo, ordenadores, periféricos)
- Perceptivo: Mejora del programa, se incluyen nuevas capacidades, nuevas prestaciones, se realizan cambios por motivos legales, administrativos, funcionales, etc.

Muy importante: Para realizar un correcto mantenimiento cuando una aplicación está instalada en múltiples clientes, es fundamental realizar un control exhaustivo de las versiones y parches.

Actualmente más del 80 % de desarrollo de software es labor de mantenimiento.

En los proyectos reales el proceso de construcción de una aplicación no suele seguir un orden secuencial sino que se producen alteraciones y vueltas atrás que dificultan y retrasan los proyectos.

Ejemplos:

- Durante la fase de prueba se llega a la conclusión que el diseño es erróneo.
- Una vez la aplicación entregada al cliente, no funciona tal como se esperaba y precisa ampliaciones y cambios que no se habían tenido en cuenta en el análisis.

ERRORES EN EL DESARROLLO DE UNA APLICACIÓN

Clasificación:

- Compilación: Errores sintácticos o semánticos en la codificación de un programa. (Son fáciles de detectar y corregir: los detecta automáticamente el compilador.)

- Ejecución (Fácil de detectar –> Parada anormal del programa
  Ejemplos: División por cero, desbordamiento de pila, bloqueo del programa, violación de segmento.

- Diseño o de Lógica
  Error en el algoritmo está mal construido y produce resultados no esperados o no funciona tal como pensamos. Pueden ser difíciles de detectar y corregir. Un error de lógica o de diseño puede provocar un error de ejecución.

- Error de análisis o especificaciones
  Si se detecta al final puede ser muy difícil de corregir, implica grandes cambios en el conjunto de la aplicación

Problemas de depurar / corregir errores:

- Distancia entre el síntoma y la causa del error
- Casos raros poco habituales
- Fallos Intermitentes
- Fallos no reproducibles
- Entorno complejo difícil de identificar el causante:

¿Qué está fallando?

Los sistemas informáticos son sistemas complejos lo que provoca que puedan surgir fallos o errores en distintos niveles: hardware, software, Sistema Operativo, la base de datos, la red, nuestra aplicación, es un error del usuario…

Importancia de la documentación de los proyectos

La documentación es fundamental en cualquier proyecto siendo imprescindible para realizar una correcta labor de mantenimiento. Todo debe estar correctamente documentado: desde los programas hasta el manejo de la aplicación por parte del usuario.

La Documentación se suele agrupar en:

- **Interna** para la empresa de desarrollo: Documentos de análisis, diseño, pruebas, programas fuentes, etc.

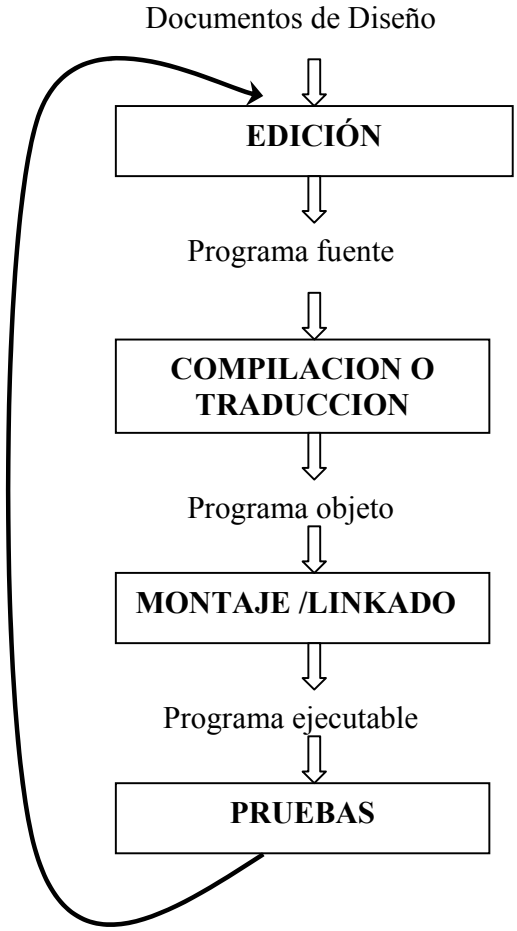
- **Externa** para el cliente: Manuales de instalación, administración, usuario, operador.

El programa fuente es un documento muy importante, es fundamenta incluir comentarios que aclaren su funcionamiento. A veces es el único documento que tenemos para realizar la tarea de mantenimiento. Hay que elaborar los programas pensando que más pronto o más tarde tendrán que ser modificados. Un buen estilo en la codificación de los programas repercute en la calidad de todo el desarrollo.

ELABORACIÓN DE PROGRAMAS

Fases en la codificación de un programa

- **Edición**
  Se realiza el fichero fuente a partir de los documentos de análisis y diseño utilizado un editor de textos.
- **Compilación / Traducción**
  Se genera automáticamente un fichero objeto (en código máquina) a partir del programa fuente mediante un programa compilador.
- **Montaje, (Linkado)**
  Se enlazan automáticamente todos los módulos, subprogramas y librerías que forman la ampliación completa, generándose el fichero ejecutable mediante un programa enlazador o linker.
- **Pruebas**
  Se realizan las pruebas necesarias que garanticen el correcto funcionamiento del programa.



Práctica: Editar un pequeño programa y realizar mediante comandos cada una de las fases, comprobar los ficheros que se generan. Generar código ensamblador. Abrir el programa con un depurador que muestre el código maquina.

LENGUAJES DE PROGRAMACIÓN

Lenguaje de programación: lenguaje artificial especialmente diseñado para indicar las instrucciones que debe realizar un ordenador. (Intermedio entre el lenguaje máquina y el lenguaje natural)

Diferencias frente al lenguaje natural: tienen normas léxicas, sintácticas y semánticas muy rígidas, no permiten la ambigüedad, ni dobles sentidos, son poco sensibles al contexto, poca información preestablecida.

Ideal: que el lenguaje de programación fuera como un lenguaje natural.
Problema: Los programadores no serían necesarios.

TIPOS DE LENGUAJES DE PROGRAMACIÓN

LENGUAJES DE BAJO NIVEL : Código máquina y ensamblador.

(Cercanos al ordenador)

Lenguajes de código máquina

Conjunto de instrucciones en binario que el ordenador es capaz de ejecutar directamente, específicos de cada tipo de procesador o CPU.

Formato típico:

Código de operación	Dato con el que operar
0100100	000000000001010

Significado: sumar al acumulador el contenido de la dirección de memoria 10.

Tipos de instrucciones:
- Mover datos
- Realizar operaciones aritméticas
- Comparar valores
- Realizar salto o cambios en el orden de ejecución de la secuencia del programa.

Ventajas
- Acceso y control absoluto de todos los recursos disponibles del ordenador
- Permite obtener la máxima velocidad y eficiencia

Inconvenientes
- Difícil de entender, Implica conocer a fondo la arquitectura y funcionamiento del Ordenador
- Difícil de codificar, corregir (Depurar), Modificar

- Se generan programas poco portables (Sólo funcionan en un tipo de procesador o arquitectura específico)

Lenguaje ensamblador

Versión más manejable para la personas del lenguaje en código máquina. Cada instrucción en código máquina tiene su equivalente en ensamblador. Es necesario disponer de un programa que traduzca el programa ensamblador a código máquina, el único que entiende realmente el procesador.

Ej- 01001000 1111 1110 -> INC R1 -- Incrementar en uno el registro interno R1

Es más fácil de entender y depurar, y tan eficiente como el código máquina al que ha sustituido por completo, aunque sigue siendo muy complejo y dependiente de la máquina.

El ensamblador sólo se utiliza en ámbitos muy limitados, como en el diseño de drivers (programas de control de dispositivos hardware), rutinas de diseño gráfico, en módulos de sistemas operativos dependientes de la arquitectura del procesador, etc.

LENGUAJES DE ALTO NIVEL

- Más próximos al lenguaje natural, cercanos al problema, nos olvidamos de la estructura interna del ordenador

Ej – No nos importa la representación interna de los números enteros (byte primer peso o menor peso)

Una Instrucción de Alto nivel -> produce decenas o cientos de instrucciones en código máquina.

Ventajas
+ Fácil de entender
+ Fácil de programar, corregir, modificar
+ Mayor productividad del programador
Permite realizar programas portables: (El mismo programa pude funcionar en distintas máquinas, Sistemas Operativos, configuraciones, etc)

Inconvenientes:
Uso menos eficientes de los recursos del ordenador, más lentos y precisan más recursos

Para que un programa escrito en alto nivel pueda ser ejecutado directamente por el ordenador debe ser traducido a código máquina o interpretado por un programa que analice cada instrucción.

GENERACIONES DE LENGUAJES DE PROGRAMACIÓN

0º : No hay programas ni lenguajes como tales, cada programa implica cambiar el cableado o la estructura interna del ordenador

1º Generación (50)
Código máquina – Ensambladores

2º Generación (60)
Basic, Cobol, Fortran (Primeros lenguajes de alto nivel, no estructurados)

3º Generación (70)
Lenguajes estructurados: Algol, Pascal, C, ADA, Modula
Específicos : Lisp, Prolog, Smalltalk

4º Generación (80)
Lenguajes declarativos :SQL
Generadores de aplicaciones, Herramientas CASE
Programación visual (VisualBasic, Visual C)
Orientados a Objeto C++, Java, Eiffel

Lenguajes según el tipo de aplicaciones

Software de sistemas: C, ADA
Software de gestión: Cobol, RPG, SQL, VisualBasic
Inteligencia artificial: Lisp, Prolog, Smalltalk
Cálculo científico: Fortran, Pascal
Desarrollo de aplicaciones en Internet: Java, C#, ASP, PHP, JSP, Perl, Javascript

Otros: Control de robot, CAD/CAM, simulación, presentaciones multimedia.(flash)

Existen miles de lenguajes de programación tanto específicos o como de propósito general.

TRADUCTORES E INTERPRETES

Un programa de alto nivel no es directamente ejecutable por el ordenador, necesita una fase intermedia de traducción o interpretación. Normalmente cada lenguaje sólo pueden ser traducido (Ej.- C) y o interpretado (Ej.- HTML), aunque existen algunos que pueden funcionar de las dos maneras o que incluso necesitan primero ser compilados y después ser interpretados (Ej.- Java)

(Similitud: con la traducción e interpretación de idioma a otro)

Traducción o Compilación:
Consiste en generar un programa de bajo nivel, generalmente código máquina, a partir de un programa escrito en lenguaje de alto nivel.

Ej- programa.c -> programa.exe

Cada instrucción en alto nivel genera múltiples instrucciones de bajo nivel.
El compilador genera un nuevo archivo con el programa traducido, pero no lo ejecuta.

Interprete

Analiza cada instrucción del programa fuente, la ejecuta y vuelve hacer lo mismo con la siguiente instrucción. No genera un archivo resultado de la interpretación.

- La ejecución de un programa interpretado es más lenta, ya que primero se debe analizar cada instrucción y después se ejecuta, mientras que un programa compilado se ejecuta directamente.
- Los lenguajes interpretados se suelen considerar más fáciles de depurar y corregir. Permiten cambiar sobre la marcha.

ESTRUCTURA DE UN PROGRAMA

Programas = Algoritmos + Estructuras de Datos

"Según: D. Niclaus Wirth"

ALGORITMO :

Conjunto ordenado de pasos a seguir que nos van permitir resolver un determinado problema.

Características que debe cumplir cualquier algoritmo:

- Correcto (Nos resuelve el problema)
- Finito (Conduce a la solución en un tiempo dado)
- Flexible (No es exclusivo a para un tipo de problema sino que sirve como método general)
- Claro (comprensible por otras personas)
- Eficiente (ahorro de tiempo y recursos)
- Portable (Independiente de la máquina o del lenguaje utilizado)

Ejemplos:

Rellenar la declaración de la renta
Efectuar una llamada telefónica
Rellenar una quiniela
Arrancar un vehículo
Subir en un ascensor

Ejemplos de Algoritmos:

Ejemplo 1 : Un cliente realiza una compra en una tienda con su tarjeta VISA, la tienda examina en el banco correspondiente si el cliente es solvente, en cuyo caso acepta la compra y en caso contrario rechaza la compra.

Solución:
1. Inicio.
2. Leer el número de la VISA.
3. Llamar al banco.
4. Comprobar con el banco si el cliente es solvente.
5. Si el cliente no es solvente se le rechaza la compra y vamos al paso 9.
6. Si el cliente es solvente se acepta la compra por tarjeta VISA.
7. Se carga el importe de la compra en la cuenta del cliente.
8. Se abona el importe de la compra en la cuenta de la tienda.
9. Fin.

Ejemplo 2 : Realizar el algoritmo que describa el proceso de ponerse los zapatos.
Solución incorrecta por no cumplir las características principales de los algoritmos:

1. Inicio.
2. Coger los zapatos.
3. Atarse los cordones del zapato izquierdo.
4. Sentarse en una silla.
5. Meterse un zapato en el pie derecho.
6. Atarse los cordones del zapato izquierdo.
7. Meterse el otro zapato en el pie izquierdo.
8. Atarse los cordones del zapato izquierdo.
9. Si los cordones de los zapatos están anudados se desatarán.
10. Volver al paso 2.
11. Fin.

Solución correcta:

1. Inicio.
2. Coger los zapatos.
3. Si los cordones de los zapatos están anudados se desatarán.
4. Sentarse en una silla.
5. Ponerse el zapato derecho en el pie derecho.
6. Atarse los cordones del zapato derecho.
7. Ponerse el zapato izquierdo en el pie izquierdo.
8. Atarse los cordones del zapato izquierdo.
9. Fin.

ESTRUCTURAS DE DATOS:

Conjunto de datos organizados relacionados entre sí definidos en un programa informático.

Dato: Un elemento que guarda información sobre un fenómeno determinado
Propiedades:

- Identificador (Nombre o denominación del dato utilizado dentro del programa).
- Tipo de dato
  - Clase de información que contiene: letras, números, valores lógicos, etc.
  - Operaciones que pueden realizar sobre ellos (Sumar, comparar, invertir)
  - Representación interna (Codificación binaria de la información)

Ejemplos:
Un número entero / El nombre de una persona / un fichero de texto / un enlace a un recurso de Internet / la Edad, una nómina, un fichero de música.

Más ejemplos:

Lista de alumnos
Un conjunto número enteros
Resultados las quinielas
Números de teléfono
Elementos de un dibujo de CAD.
Estado civil
Historial clínico de un enfermo

CLASIFICACIÓN DE LAS ESTRUCTURAS DE DATOS / TIPOS DE DATOS

.

Clasificaciones generales:

a)
Constantes: No cambian de valor.
Variables: Alteran su valor a lo largo de la ejecución de programa.

b)
Simples: Almacenan un solo dato.
Compuestos: Almacenan varios datos.

b)
Estáticas: Tiene un tamaño y estructura predefinido que no varía durante la ejecución.
Dinámicas: Cambian de tamaña o estructura durante la ejecución del programa.

c) Interna: Se almacena en memoria principal (RAM)
Externa: Se almacena en memoria secundaria (Disco)

Datos Simples

Ordinales

Entero (1,2,2003,12938,-9,0)

Carácter (‘a’,’B’,’/’,’+’)

Lógico (Verdadero, falso)

Enumerado (Colores de un semáforo, fichas de ajedrez, días de la semana)

Reales

(-0.234, 23004.34, 0.023 x 10^{-23},1/7)

Datos Compuestos

Estáticos

Tabla (Vector, Matriz, Poliedro)

Número finito de valores de un mismo tipo, almacenados consecutivamente.

Registro

Número finito de valores de distintos tipos.

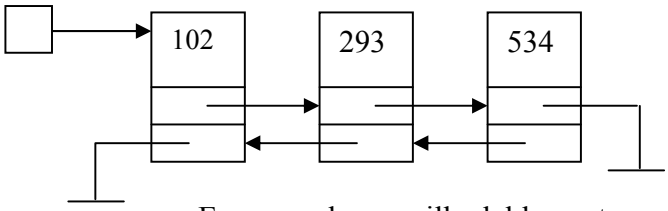
Conjunto

Número finito de valores de un mismo tipo donde no importa el orden o la disposición de los mismo

Dinámicas

Listas (Cada elemento tiene un antecesor y un sucesor)

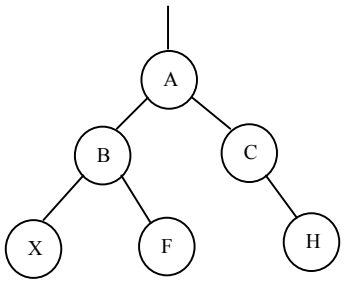
Tipos: Colas, Pilas, Anillos



Esquema de un anillo doblemente encadenado.

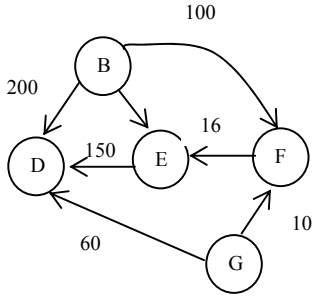
Árboles (Cada elemento tiene un antecesor y N sucesores)

Tipos: Binarios, ordenador, Árboles B



Grafos (Cada elemento tiene N antecesores y M sucesor)

Tipos: Dirigidos, etiquetados, conexos…



Características:

- Varían en capacidad y estructura a lo largo de la ejecución.
- Mas versátiles, pero más difíciles de programar
- No desperdician recursos de memoria, se adaptan a la naturaleza de programa

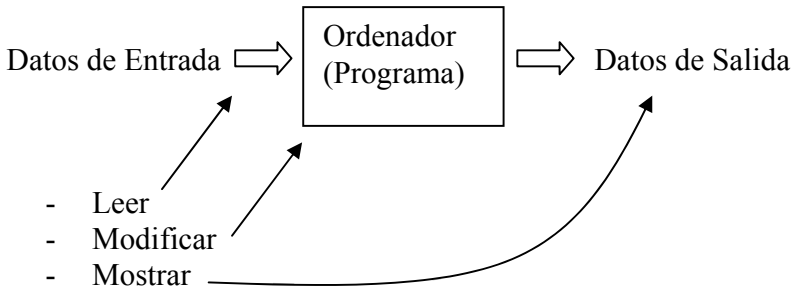
Estructuras externas

- Ficheros
  - Planos :
    - Texto
    - Binarios: imágenes, sonidos, ejecutables, etc,
  - Registros: Clientes, nóminas
- Bases de Datos

- Más capacidad, pero más lentas

Pensar un ejemplo para cada estructura

OPERACIONES SOBRE LOS DATOS



Ej.- Sumar dos números y mostrar el resultado.
Modificar el saldo de un cliente de nuestra base de datos.

Operadores

La manipulación de los datos de entrada que realiza un algoritmo se hace aplicando sobre los mismos una o más operaciones. Muchas operaciones que se realizar con los datos (variables o constantes) se representan mediante los operadores.
Sólo se aplican sobre determinados tipos de datos (no podemos sumar manzanas con retratos), su significado puede variar. No se permite mezclar datos de distintos tipos.
Cada lenguaje suele definir distintos tipos de operadores

- Operadores aritméticos (Matemáticos)

Suma (+) Resta (-) Multiplicación (*) División (/) Potencia (**, ^), Módulo (%,mod)
Signo (-/+)

¡Ojo! Operaciones con enteros y con reales.
Resultado → un número (Entero, Real)

- Operadores Relacionales (Comparación)

Si Izquierda es Mayor > que Derecha
Igual (=) Distinto (≠) Mayor (>) Menor (<) Mayor y igual (>=) Menor y igual (<=)

- Comparición de cifras: Dato > 100, Numero < (4 * Cifra)
- Comparación de caracteres y cadenas:
  'a' < 'z' y 'Z' < 'z'

  "ANA" < "LUIS" < "Luis" < "ana" < "luis"

  Nota: ¡Ojo con la ñ y acentos¡, las minúsculas están después que las mayúsculas. Consultar tabla ASCII

Resultado → (Verdadero o Falso)

- Operadores Lógicos
  Conjunción Y (AND, &&) Disyunción O (OR, ||) Negación NO (NOT, ¡)

Resultado → (Verdadero o Falso)

Tablas de Verdad

A	NOT A
V	F
F	V

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

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

Equivalencia entre expresiones lógicas

NOT (A AND B) = NOT(A) OR NOT(B)
NOT (A OR B) = NOT(A) AND NOT (B)
NOT (NOT A) = (A)
NOT (A > B) = (A <= B)
NOT (A >= B) = (A < B)
NOT (A = B) = (A ≠ B)

Orden de evaluación de los operadores:

Depende del lenguaje: Siempre de Izquierda a Derecha

1. Paréntesis
2. Signo
3. Negación
4. Potencia
5. Producto, división o módulo
6. Suma o resta
7. Relacionales
8. Lógicos

Puede de ser un lío

3 – 5 * 8 / 2 /10 * –2 > 10** 2 y 5 * 2 > 2

Lo mejor utilizar paréntesis

A > 23 y (B+ C) < D / 2