Una mica d'història.

- Charles Babagge (s XIX) prediu la màquina analítica on es defineixen les teories en que es basen els ordinadors actuals.
- La seva colaboradra, Ada Lovedby, realitzà els primers programes per aquella suposada màquina.
- El diseny de Babagge defineix 5 unitats bàsiques per la màquina analítica.

Les unitats bàsiques de la màquina analítica.

1) Unitat d'entrada
2) Memòria
3) Unitat de Control
4) Unitat aritmètica-lógica
5) Unitat de Sortida.

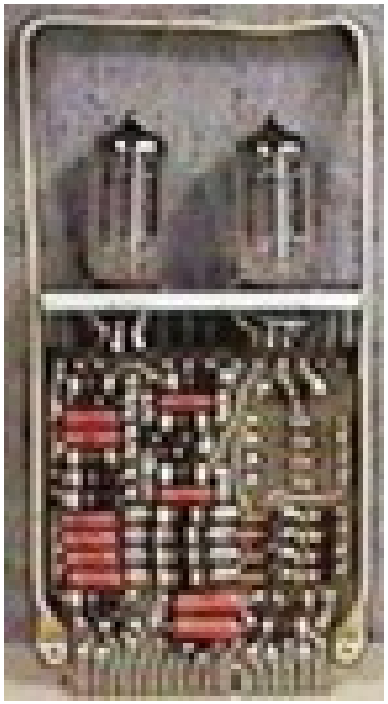
Les màquines actuals segueixen aquests principis bàsics dissenyats per Babagge.

Evolució dels ordinadors

- Generalment es parla de 5 generacions d'ordinadors segons la seva arquitectura i la forma com es relacionen amb nosaltres.
- Als anys 40 apareixen les primeres màquines (Mark I, ENIAC, EDVAC, UNIVAC) de grans dimensions i gran consum elèctric
  - Mark I feia servir 800 km de cable
  - Univac pesava uns 7.200 kg.

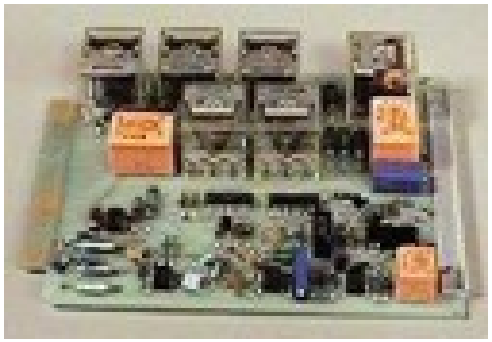
1a Generació

- La primera generació son màquines que es caracteritzaven per
  - Estar basades en tubs de buit.
  - Programades en llenguatge màquina.
  - Dades i programes s'introduïen fent servir targetes perforades.
  - Fan servir cilindres magnètics per emmagatzemar informació i instruccions internes.
  - Us del sistema binari per representar dades.
  - 1951 UNIVAC, primer computador comercial.



2a Generació

- Als anys 60 comencen a implementar-se les primeres màquines basades en circuits de transistors.
- Es programen amb llenguatges d'alt nivell.
- Programes fets a mides per un equip d'experts. Cal saber programar per obtenir resultats.

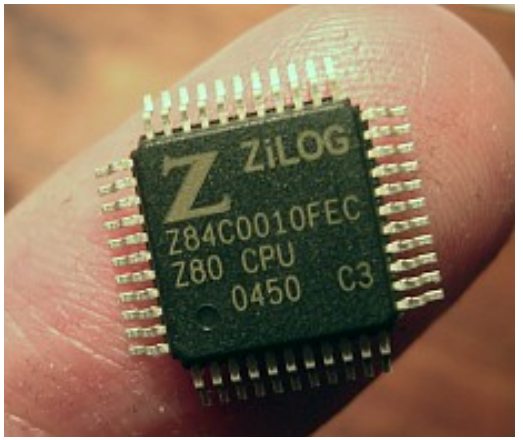


3a Generació

- L'any 1964 apareix la IBM 360, que fa servir circuits integrats
- Els xips permeten enmagatzemar i processar la informació, guardant la informació com a càrregues elèctriques.

4a Generació

- Ordinadors basats en microprocessadors.
- Un sol chip pot fer diverses funcions, com ara ser la unitat de control i la ALU.
- Apareixen els xips de memòria.
- Desenvolupament dels microordinadors
  - 1972 Steve Wozniak y Steve Jobs treuen al mercat el primer microordinador d'us massiu.



5a Generació

- La velocitat en la que evoluciona el hardware es molt més elevada que la del software. Hi ha iniciatives per tal de dur el desenvolupament del software a una velòcitat similar.
  - Fer servir instruccions més naturals per comunicar-nos amb els ordinadors.
- L'any 1983 Japó inicia el “programa de la cinquena generació de computadors” El seu objectiu era desenvolupar una nova clase d'ordinadors que, fent servir tècniques d'intel·ligència artifical, fossin capaces de resoldre problemes complexos, com ara la traducció automàtica entre dos llengües naturals.

Llenguatges de programació.

- Els ordinadors son màquines amb un llenguatge especific
- Han de interpretar les instruccions que reben.
- Els llenguatges de programació ens permeten realitzar la comunicació home-màquina.

- Els llenguatges de programació ens permeten
  - Escriure operacions per resoldre problemes
    - Algorismes
  - Traduir les expressions dels algoritmes a llenguatge màquina (binari) -> compilar.
- Els ordinadors son per tant unes màquines que poden resoldre aquells problemes que podem expressar mitjançant un algorisme

- Tenim gran quantitat de llenguatges de programació, cadascun d'ells amb les seves sintaxis, gramàtica i terminologia.
- Tenim llenguatges orientats a
  - Aplicacions científiques (Fortran)
  - Àmbit empresarial (Cobol)
  - Propòsit múltiple (C)
  - Entorns Web (PHP)
  - ...

Llenguatges de programació

Segons el seu nivell d'abstracció els podem classificar en

Baix Nivell

El seu funcionament es proper al de un ordinador.

Codi Màquina o Ensamblador son exemples d'aquests tipus.

Alta dependència amb el Hardware.

Alt Nivell

Sentències comprensibles per les persones.

Portables.

Cal realitzar un proces per tal que l'ordinador els entengui (compilació o interpretació)

Llenguatges de programació

Segons com s'executen podem parlar de

- **Llenguatges compilats**. El **Codi Font** del programa, un cop finalitzat, cal que sigui traduït a un idioma comprensible per l'ordinador (**codi objecte**). Es fa servir una aplicació, anomenada compilador per fer aquesta traducció.
- **Llenguatges interpretats**. La traducció es fa durant la execució del programa.

Llenguatges de programació

Podem classificar també els llenguatges segons com treballen i com han estat concebuts.

- **Llenguatges Imperatius**. Fan servir instruccions com a unitat de treball dels programes (Cobol, Pascal, C, ...)
- **Llenguatges declaratius**. Es constitueixen mitjançant descripcions de funcions o expressions lògiques (Lisp, Prolog). Indiquem que el que es vol obtenir.
- **Llenguatges orientats a objectes**. El seu disseny es basa en dades i estructures. L'untat de proces es l'objecte, que inclou tant les dades com les operacions que actuen sobre ells (Smalltalk, C++, Java).
- **Llenguatges orientats al problema**. Dissenyats per problemes específics, normalment de gestió, acostumen a ser generadors d'aplicacions.
- **Llenguatges naturals**. Busquen aproximar el disseny i la construcció dels programes al llenguatge de les persones

Llenguatges de programació

Segons la seva evolució històrica, molt relacionada amb la evolució del maquinari, parlem de llenguatges

- **De primera generació**. Codi màquina.
- **De segona generació**. Ensamblador.
- **De tercera generació**. Llenguatges Imperatius (Cobol, Fortran, Pascal, C, ...)
- **De quarta generació**. Engloba tant els llenguatges orientats a objectes (Java, ...) com aquells que estan dissenyats per obtenir informació d'eines o bases de dades (Natural, SQL) connectant peces prefabricades.
- **De cinquena generació**. Orientats a la intel·ligència artificial.

Cicle de vida del software.

Descriu el desenvolupament del software, des de les fases inicials fins a les finals.

Permet marcar una serie de punts de validació durant el desenvolupament de la aplicació.

Un desenvolupament d'aplicacions presenta 7 fases

1) Presa de requeriments.
2) Especificació
3) Arquitectura
4) Programació
5) Testeig
6) Documentació
7) Manteniment

Cicle de vida del software.

Presa de requeriments.

- Cal determinar quines funcionalitats ha de tenir la aplicació que hem de desenvolupar.
- També ens servirà per saber quins recursos de software / hardware podem fer servir.
- Caldrà fer entrevistes amb usuaris finals de la aplicació per conèixer les seves necessitats.
- En ocasions ens serà útil l'us d'un prototipus.
- Cal identificar els interlocutors i quin paper juguen.

Cicle de vida del software.

Presa de requeriments - Problemes.

- Els usuaris no tenen clar que volen, o no s'involucren en el procès.
- Els analistes no coneixen el negoci.
- Intentar encaixar el sistema en un model existent, quan pot tenir menys cost començar un sistema nou.

Cicle de vida del software.

Especificació.

- Descriu el comportament del software un cop desenvolupat.
- Cal identificar les necessitats del negoci i interactuar amb els usuaris funcionals per prioritzar, classificar, identificar i especificar els requeriments de software.
- Una eina molt útil son els Casos d'Us.

Cicle de vida del software.

Arquitectura.

- Disseny d'alt nivell de la estructura de un sistema.
- Fa una descripció a nivell general de com serà el sistema
- Es dissenyen els components d'una aplicació en funció als requeriments i la especificació.
- Es generen
    - Diagrames de Classe.
    - Diagrames de Base de dades.
    - Diagrama de Desplegament.
    - Diagrama de Sequència.

Cicle de vida del software.

Programació.

- És la fase més obvia del desenvolupament del sistema, en ella es genera el codi segons la documentació generada durant la fase d'arquitectura..

Cicle de vida del software.

Testeig.

- Cal verificar que el software realitzi correctament les tasques especificades.
- Els diferents mòduls els verificarem per separat, i un cop verificats es faran proves de conjunt.
- Si fos necessari caldrà també veure com s'integra el nostre software amb altres softwares existents.
- Es recomana que les proves les facin persones diferents als desenvolupadors.

Cicle de vida del software.

Documentació

- Durant el cicle de vida, haurem generat ja documentació funcional o tècnica del software (casos d'us, diagrames de classes, ...)
- Cal generar tota la documentació necessària per els usuaris
    - Manuals de l'aplicació.
    - Material per la formació.
- Cal també generar tota la documentació tècnica per tal de facilitar possibles desenvolupaments futurs, ampliacions, canvis, correccions, ...

Cicle de vida del software.

Manteniment.

Aquesta fase pot arribar a extendres més temps que el desenvolupament inicial del programa.

- Permet per una banda realitzar correccions a errors que ens han passat desapercebuts a la fase de testeig.
- També està pensat per incorporar nous requisits i fer evolucionar el programa.

Programació Orientada a Objectes

Paradigma de programació que fa servir els objectes i les seves interaccions per dissenyar aplicacions informàtiques.

Programació Orientada a Objectes

Clase

- Es la plantilla a partir de la qual definirem els objectes. Defineix les propietats i els mètodes dels objectes.
- Podem afirmar que la classe es la definició dels objectes.

Programació Orientada a Objectes

Objectes

- Son instancies concretes de les classes.
- Presenten una serie de característiques (Propietats) i funcionalitats (Mètodes) que es fan servir per interactuar amb altres objectes de la aplicació.
- Internament poden ser complexos, però a nosaltres ens interessarà que poden fer enlloc de com ho fa.

Programació Orientada a Objectes

Característiques de la programació orientada a objectes

- Tipificació estricta.
  - Quan tinguem dos expressions relacionades (expressions, assignacions, ...) cal que es facin sempre entre objectes del mateix tipus.
  - *No es poden sumar pomes amb peres.*

Programació Orientada a Objectes

Característiques de la programació orientada a objectes

- Encapsulament.
  - Els objectes oculten la seva informació i només donen accés a allò necessari.
  - Podem accedir a les dades del objecte a través de mètodes, el que permet verificar la informació.
  - Si cal canviar la manera com s'emmagatzema la informació a un objecte, la resta de objectes que interactuen amb ell, no es veuen afectats per aquests canvis.

Programació Orientada a Objectes

Característiques de la programació orientada a objectes

- Genericitat.
  - Es la propietat que permet definir mètodes que tenen com a paràmetres elements de qualsevol tipus.
  - Exemple. Tenir objectes que poden guardar una llista de elements de qualsevol tipus.

Programació Orientada a Objectes

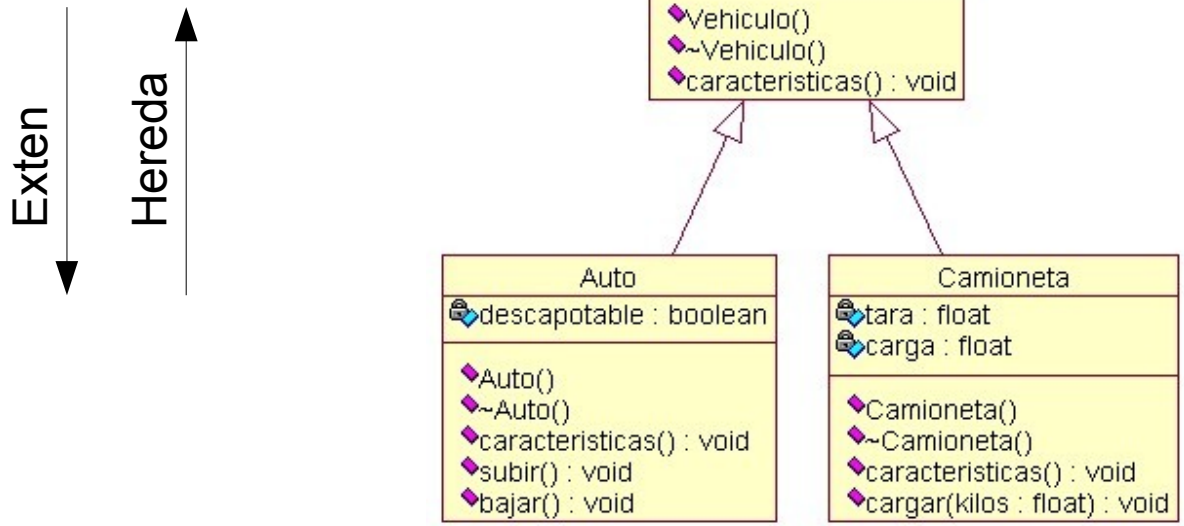
Característiques de la programació orientada a objectes

- Herència.
  - Permet que definim una classe nova a partir de una o més classes existents, fent que tingui les característiques de la classe inicial, afegint aquelles propietats i mètodes que el programador vulgui afegir-hi.
  - Creem noves classes a partir de existents, descrivint les diferències entre elles.
  - En cas que sigui necessari, es poden **sobreescriure** mètodes existents substituint la seva funcionalitat.

Programació Orientada a Objectes

Característiques de la programació orientada a objectes

- Herència.



Programació Orientada a Objectes

Característiques de la programació orientada a objectes

- Polimorfisme.
  - Comportaments associats a diferents objectes, els podem cridar amb al mateix nom.
  - Al fer la crida a aquesta funcionalitat, es farà servir el funcionament referent al objecte involucrat
  - Una operació pot fer tasques diferents, depenent de l'objecte sobre el que s'invoca.
  - Exemple. Podem definir una operació Suma que, internament, operarà segon el tipus de objecte que tinguem

Factors de Qualitat

- Factors de qualitat interns
  - que es relacionen amb la qualitat del codi
- Factors de qualitat externs
  - que son avaluables per els usuaris finals

Factors de Qualitat Externs

Reutilització del Codi.

- Permet tornar a fer servir parts de codi creats per altres aplicacions a la nostra aplicació.
- Permet que codi fet a la nostra aplicació es pugui fer servir en futures aplicacions.
- La programació Orientada a Objectes facilita la reutilització dons les classes son fàcilment portables.

Factors de Qualitat Externs

Escalabilitat de la aplicació.

- Permet que una aplicació s'amplii i adapti a canvis en les seves especificacions.
- Quan més gran es l'aplicació es mes difícil mantenir la seva escalabilitat. Cal
  - Simplificar el disseny
  - Descentralització, fent servir mòduls autònoms

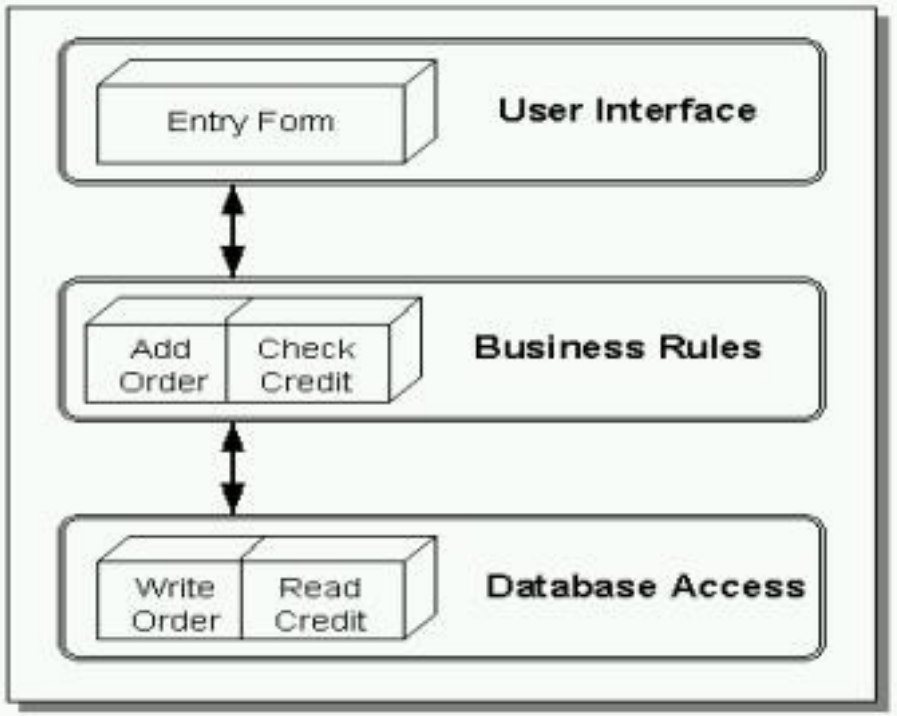
Factors de Qualitat Externs

Portabilitat

- Facilitat de usar una mateixa aplicació en diferents plataformes de programari i maquinari.
- Les incompatibilitats venen per les característiques del sistema i no pas per la lògica de la aplicació.
- Es sol buscar arquitectures en capes

Factors de Qualitat Externs

Portabilitat



Factors de Qualitat Externs

Usabilitat

- Cal que persones de orígens i formació diferents sigui capaç de familiaritzar de manera ràpida amb la aplicació.
- Cal que els usuaris tinguin la sensació que es fàcil treballar amb la aplicació.

Factors de Qualitat Externs

Funcionalitat

- Poder ampliar les tasques que son capaces de fer.
- Perills al ampliar les funcionalitats
  - Canviar la manera de accedir a les funcionalitats més usades per els usuaris, que afecta a la usabilitat.
  - Pot suposar canvis en les estructures de dades, això pot afectar a altres requisits

Factors de Qualitat Externs

Precisió

- Cal que el programa realitzi les tasques marcades a les especificacions de manera precisa.
- Problema, les especificacions varien, i els usuaris no tenen clar el que necessiten.

Factors de Qualitat Externs

Robustesa

- Una aplicació es robusta si reacciona de manera correcta a situacions anòmales.
- Cal fer aplicacions a prova de bombes.
- Si la aplicació es penja sovint, transmet males sensacions als usuaris, que buscaran alternatives al nostre programari.

Factors de Qualitat Externs

Compatibilitat

- Els components de software, no es dissenyen per usar-se de manera aïllada, sinó que cal que interactuen entre ells.
- Cal un disseny homogeni per facilitat la compatibilitat.

Factors de Qualitat Externs

Rendiment.

- Cal que els nostres programes funcionin fent servir el mínim de recursos de hardware.
- Cal donar solucions especifiques a cada problema. Un canvi pot resultar complexe, afectant a la escalabilitat

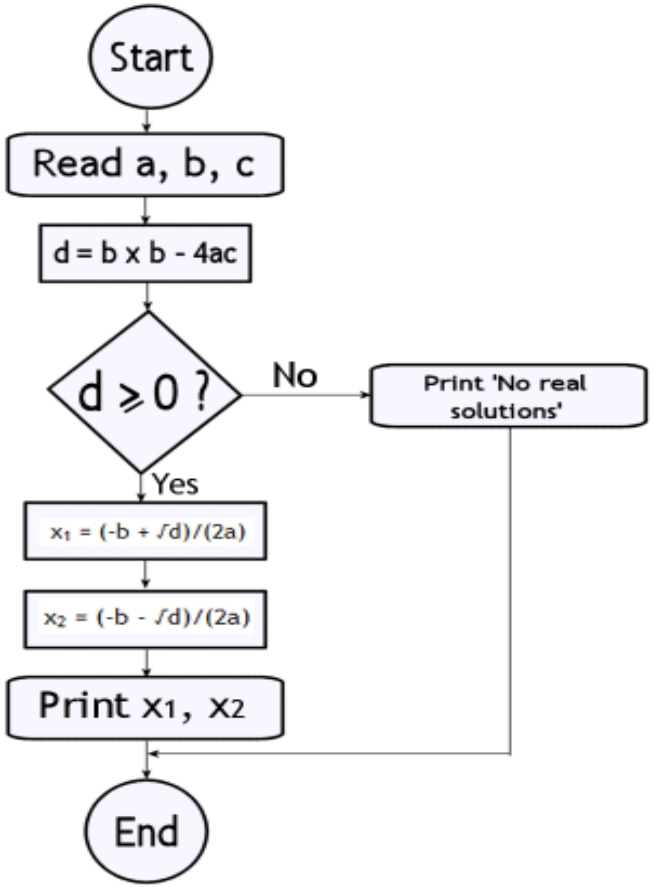
Factors de Qualitat Externs

Oportunitat.

- Donar la aplicació en el moment que els usuaris la necessiten ens assegurarà l'èxit de l'aplicació.

Algorismes

- una seqüència finita d'instruccions realitzables, no ambigües, l'execució de les quals condueix a una resolució d'un problema. (Viquipèdia).



Algorismes

Disposem de diverses eines per tal de mostrar un algorisme.

- Diagrames de flux.
- Pseudocodi.

Algorismes

Diagrama de Flux.

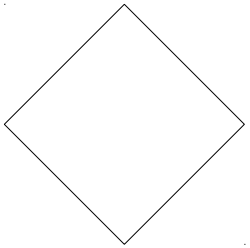
- Descripcions gràfiques d'algorismes. Fan servir símbols estàndards connectats per fletxes que indiquen la seqüència de instruccions.
- Fan servir molt espai, per tant només s'usen per algoritmes petits.
- Sempre tenen un punt de inici i un punt de finalització.

Algorismes

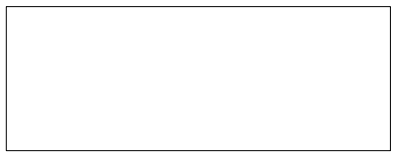
Diagrama de Flux.

- Els principals símbols que podem fer servir són.

Inici o final de diagrama



Condició (bifurcació)



Procés

Entrada o sortida de dades

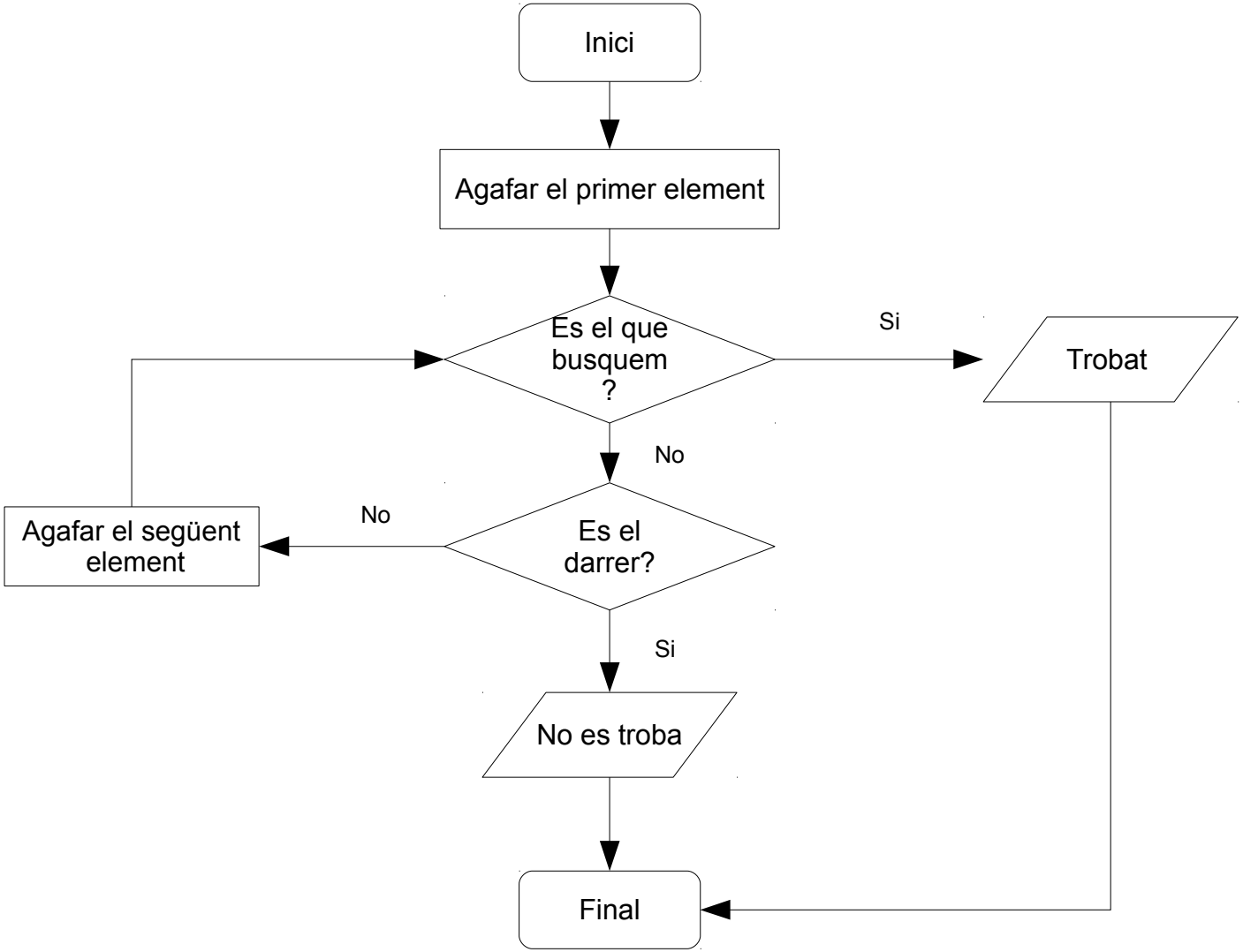
Algorismes

Pseudocodi.

- Es una descripció de alt nivell d'un algorisme que mescla llenguatge natural amb convencions sintàctiques dels llenguatges de programació.
- Pensat per que les persones entenguin un algorisme, sense tenir en compte el llenguatge en el que es desenvoluparà.

Algorismes

Exemple. Cerca seqüencial



Algorismes

Exemple. Cerca seqüencial.

1) Prenem el primer element de la llista.
2) Es el element que cerquem?
    1) ja el tenim i finalitzem la cerca.
3) Si no es l'element que es cerca
    1) Estem al darrer element
        1) No. Agafar el següent element i tornar al pas 2
        2) Si. El element no es troba a la llista i finalitzem

Algorismes

Exemple. Cerca binaria o dicotómica.

Nomes per llistes ordenades.

```
Mentre (izq<=der) Fer
  centre=(izq+der)/2 /* divisió sencera: es trunca la part decimal */
  Si (vec[centre]=dada) Llavors
    Retornar veritable /* o Retornar pos, la posició de l'element localitzat*/
  Fi Si
  Si (dada<vec[centre]) Llavors
    der=centre-1
  Sinó
    izq=centre+1
  Fi Si
Fi Mentre
Retornar Fals /* o retornar -1 segons convenccio */
```