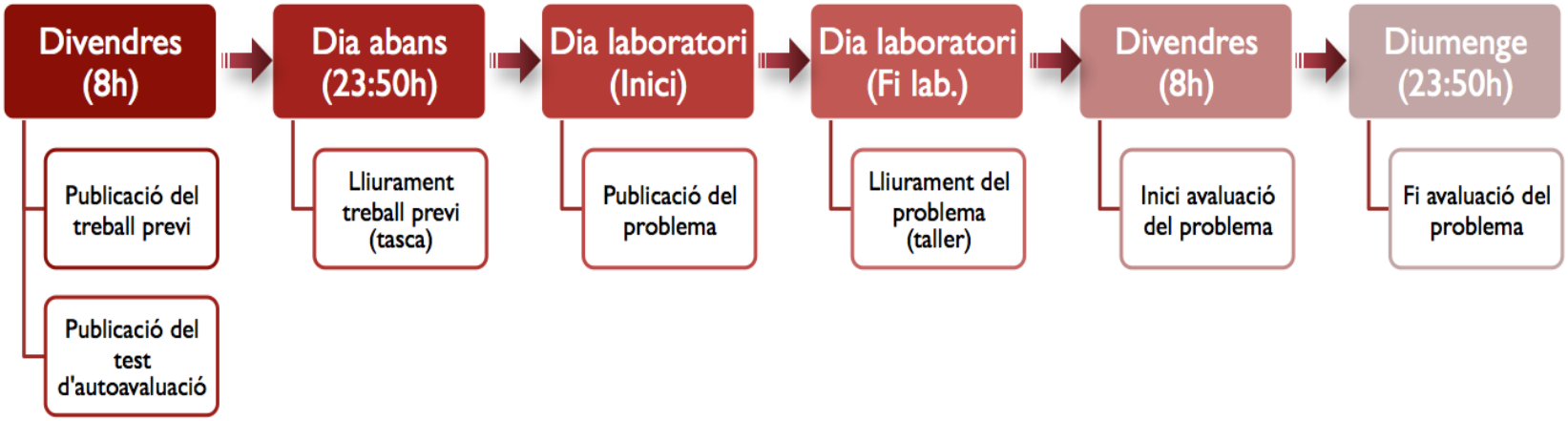
Presentació dels laboratoris

Introducció

- Pes dels laboratoris a la nota: 10%
  - Tots els laboratoris tenen el mateix pes
  - Es descarten les 3 notes més baixes
- Nota del laboratori:
  - 80% avaluació del problema lliurat al laboratori
  - 20% avaluació de l'avaluació feta (nota d'avaluador)
- Organització de les sessions:



Entorn de Programació C

Índex

- **Objectius de la sessió**
- Com es construeix un programa
- Implementació d'un programa:
  - Codificació de l'algorisme:
    - Edició del codi font en C.
  - Obtenir la versió executable i executar:
    - Procés de compilació.
    - Detecció i depuració d'errors.
    - Verificació del programa
- Exemple pràctic
- Entorn de desenvolupament: MS-DOS i Eina Code::Blocks

Objectius de la sessió

1. Entendre les etapes del procés de creació d'un programa, independentment del llenguatge de programació.
2. Conèixer les eines i passos concrets pel llenguatge que usarem a FP: C.
3. Entendre el funcionament de l'entorn de programació en C.

Índex

- Objectius de la sessió
- **Com es construeix un programa**
- Implementació d'un programa:
  - Codificació de l'algorisme:
    - Edició del codi font en C.
  - Obtenir la versió executable i executar:
    - Procés de compilació.
    - Detecció i depuració d'errors.
    - Verificació del programa
- Exemple pràctic
- Entorn de desenvolupament: MS-DOS i Eina Code::Blocks

Com es construeix un programa

Cal aplicar les fases del **cicle de vida** del programari:

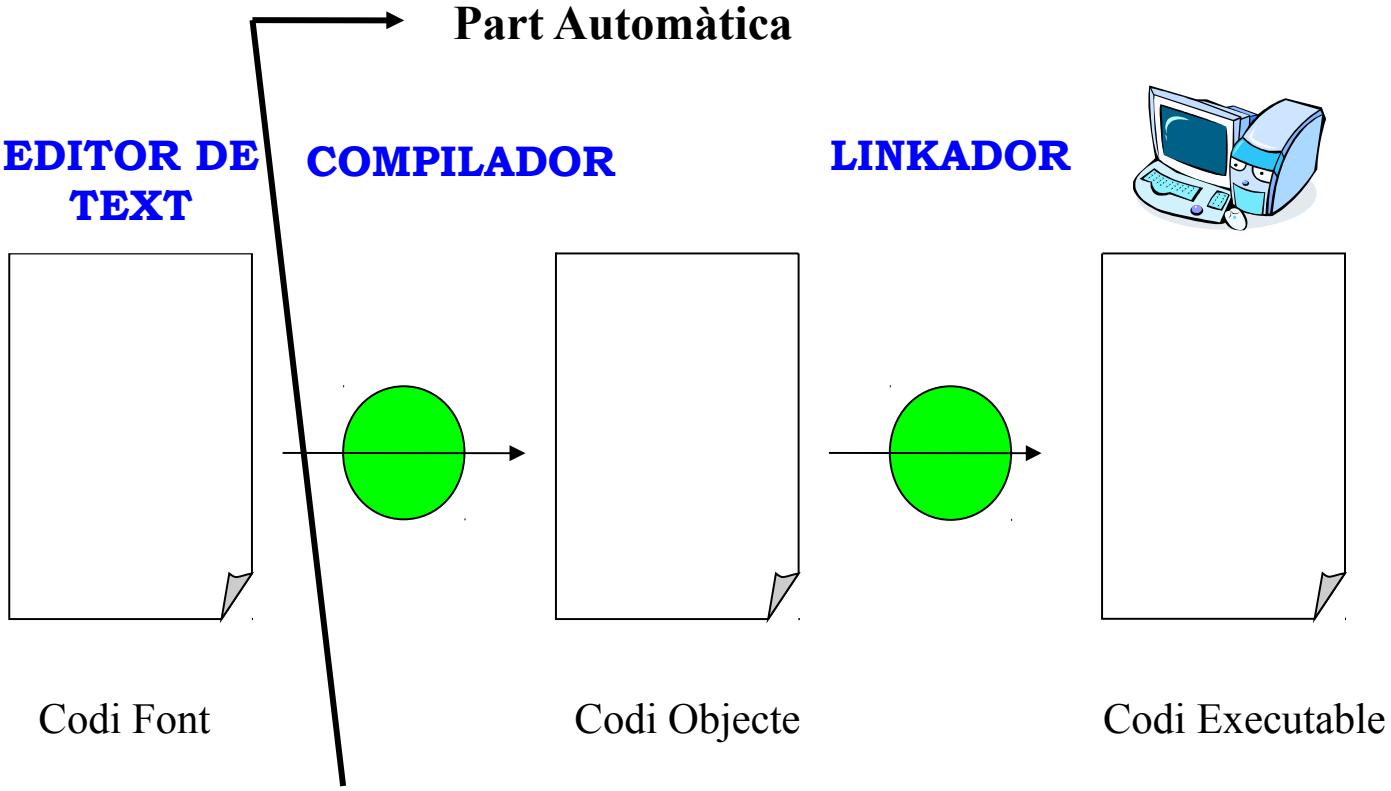
1. Identificar el problema a resoldre.
2. Analitzar les necessitats que ha de cobrir l'aplicació.
3. Dissenyar l'arquitectura de l'aplicació
4. **Implementar el programa**
5. Instal·lar i mantenir l'aplicació

Índex

- Objectius de la sessió
- Com es construeix un programa
- **Implementació d'un programa:**
  - Codificació de l'algorisme:
    - Edició del codi font en C.
  - Obtenir la versió executable i executar:
    - Procés de compilació.
    - Detecció i depuració d'errors.
    - Verificació del programa
- Exemple pràctic
- Entorn de desenvolupament: MS-DOS i Eina Code::Blocks

Implementació d'un programa

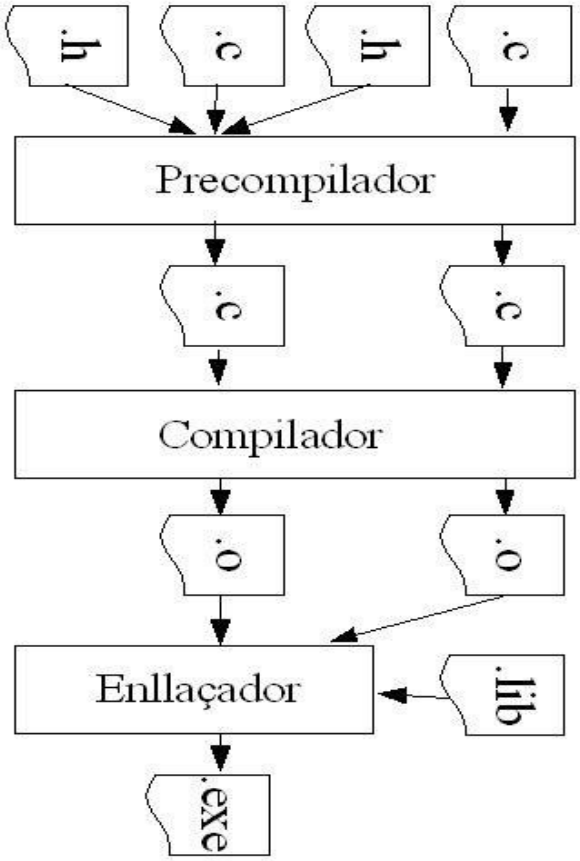
- Passos:



Codificació de l'algorisme: Edició del codi font en C

- Es pot usar qualsevol editor de text:
  - Windows: Crimson Editor, Notepad++, ...
  - Linux: gedit, vi, ...
  - MacOS X: Code, TextMate, ...
- Hi ha editors que reconeixen els llenguatges i ofereixen algunes eines d'ajuda.
- Hi ha entorns sencers de programació que permeten fer tots els passos (edició, compilació i execució) des del mateix entorn.
- Durant el curs usarem l'eina: Code::Blocks
- Farem servir eines de lliure distribució.

Obtenir la versió executable: Procés de compilació



- **Fases de la compilació:**

1. **Precompilador:** s'encarrega de modificar el codi font mitjançant una sèrie d'instruccions (directives) simplificant la feina del compilador.

2. **Compilador:** genera codi objecte a partir de codi preprocessat.

3. **Enllançador (Linker):** uneix el codi objecte del programa (o dels diferents mòduls) amb les llibreries per generar el programa executable final.

Possibles errors:

- Errors de compilació.
- Errors d'enllaçat.
- Errors d'execució.

Obtenir la versió executable: Procés de compilació

Tipus de fitxers en C:

- **Font (.c):** Fitxer amb les instruccions del llenguatge C.
- **Objecte (.o o .obj):** Són fitxers intermedis creats a partir de la compilació, entenedors per l'enllaçador.
- **Llibreria (.a o .lib):** Agrupen en un únic arxiu objecte diferents programes.
- **Capçalera(.h):** Contenen definicions i declaracions compartides per diferents fitxers font, així com les capçaleres dels programes que utilitzem.
- **Executable (.exe):** es guarda en un sol fitxer tot el codi màquina dels fitxers objectes associats al programa, ja en un format executable.

Obtenir la versió executable: Detecció i depuració d'errors

- Normalment un programa, fet per un programador novel, no funciona a la primera.
- **Incidències en la compilació:** poden ser de dos tipus:
  - *Errors*: acostumen a ser de sintaxi, quan part del codi no s'ajusta a les regles sintàctiques del llenguatge (C).
    - *Ex: deixar-nos de posar un punt i coma al final d'una instrucció.*
    - No es passa a la següent fase si aquesta no és superada pel compilador.
  - *Advertiments ("Warnings"):* informen d'errors no sintàctics que poden ser lleus, per això el procés de compilació termina i es genera l'executable.
    - *Ex: en sumar una variable entera i una variable real, el compilador donarà un advertiment.*
    - Com la gravetat de l'advertència pot en alguns casos ser més severa es aconsellable eliminar el motiu que ha generat l'advertència.

Obtenir la versió executable: Detecció i depuració d'errors

- **Errors d'enllaçat:** es produeixen en la fase d'enllaçat quan no es troben alguns dels programes que s'utilitzen en el nostre programa.
  - *Ex: no troba els fitxers de la capçalera (.h) , per exemple: <stdio.h>*
- **Errors d'execució:** es produeixen en executar el programa. S'ha compilat i enllaçat sense errors, però el programa no funciona correctament o no fa el que hauria de fer.
  - Són els errors més difícils de detectar i corregir.
  - *Ex: divisió per zero, arrel quadrada d'un negatiu.*

A vegades és útil fer servir el depurador ("*Debugger*"), que permet fer un seguiment del programa: executar línia per línia, aturar l'execució en alguna línia determinada del codi, o veure el valor de les variables en un lloc concret del programa.

Verificació del programa

Verificar el programa consisteix a comprovar que aquest soluciona el problema pel qual va ser dissenyat.

Per comprovar-ho cal elaborar un **Joc de proves**.

El joc de proves està format per una llista de situacions (proves), on en cada situació s'especifiquen les dades que s'introdueixen al programa i el resultat esperat pel programa.

- **Prova de fum**. És denomina així a la prova bàsica de funcionament.

Verificació del programa

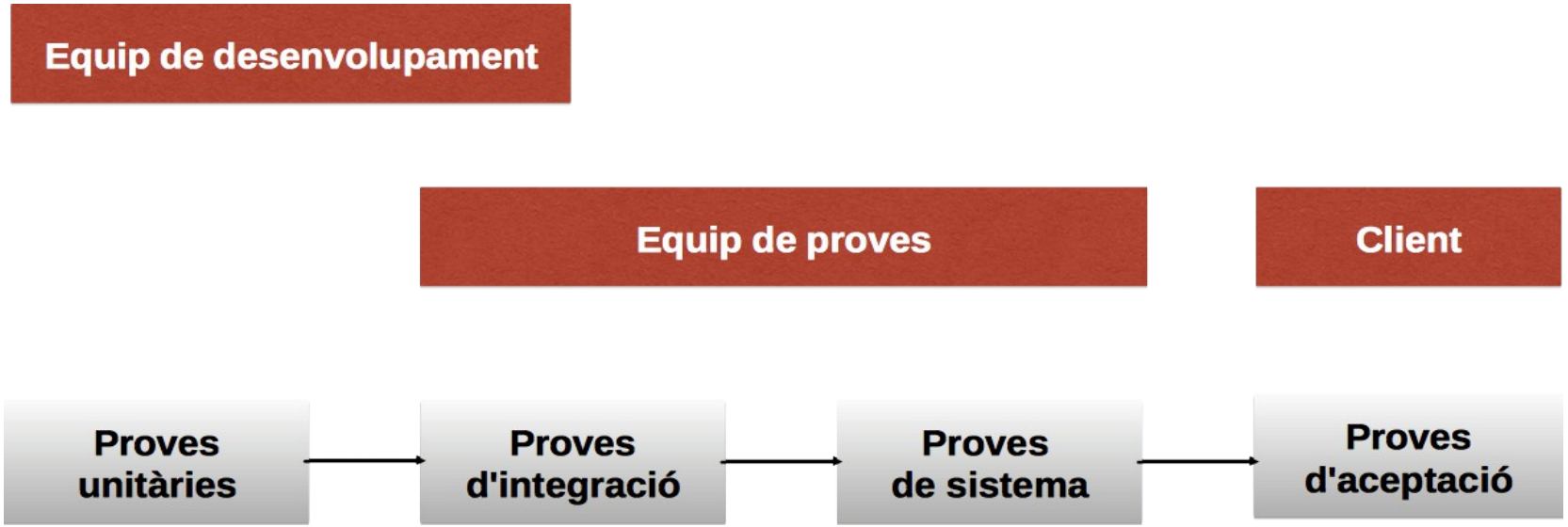
- **Què és una prova?**
  - Una prova és un conjunt d'activitats **planificades** i que es realitzen de manera **sistemàtica**.
  - Estan orientades a **verificar** i **validar** el software.
    - **Verificar**.
      Comprovar que el programa implementa correctament la funció específica.
    - **Validar**.
      Comprovar que el programa cumpleix amb els requeriments del client.

Verificació del programa

- **Característiques de les proves?**
  - **Destructives**.
    Es dissenyen per a buscar defectes i errades.
  - Ofereixen confiança sobre la qualitat del programari: els resultats són els especificats i els esperats.
  - Han de ser **reproduibles**.

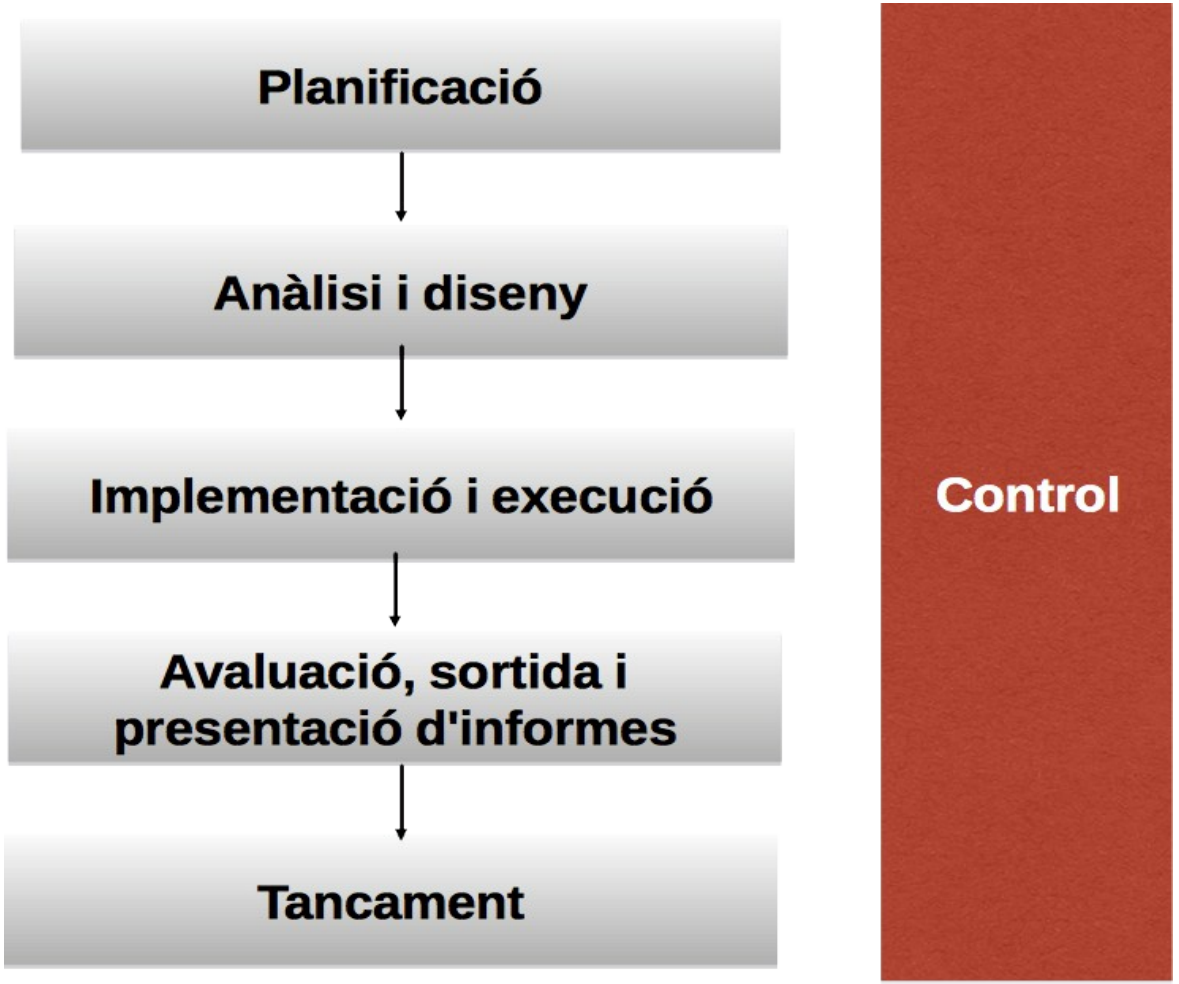
Verificació del programa

- **Nivells de les proves**



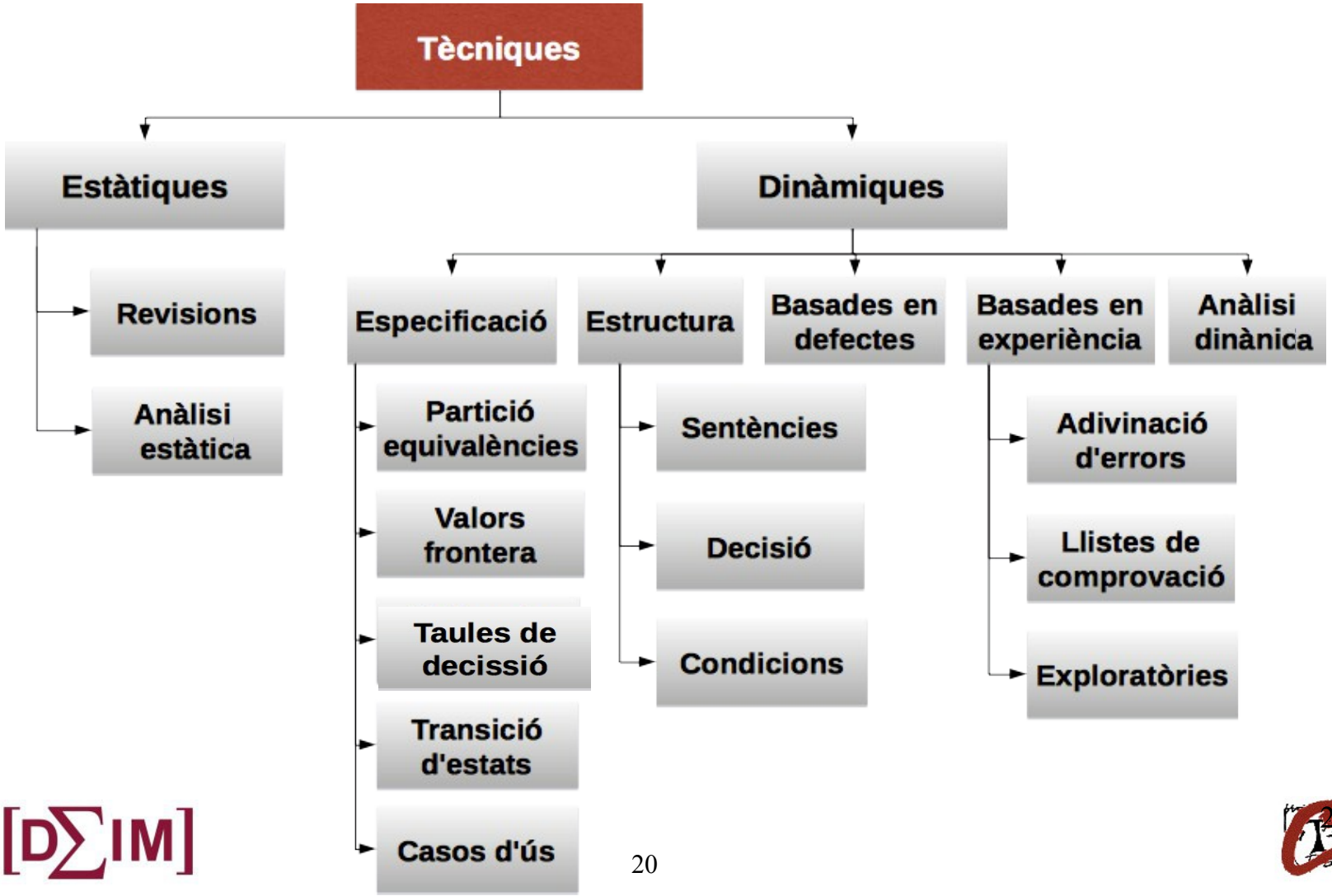
Verificació del programa

- **Procés de les proves**



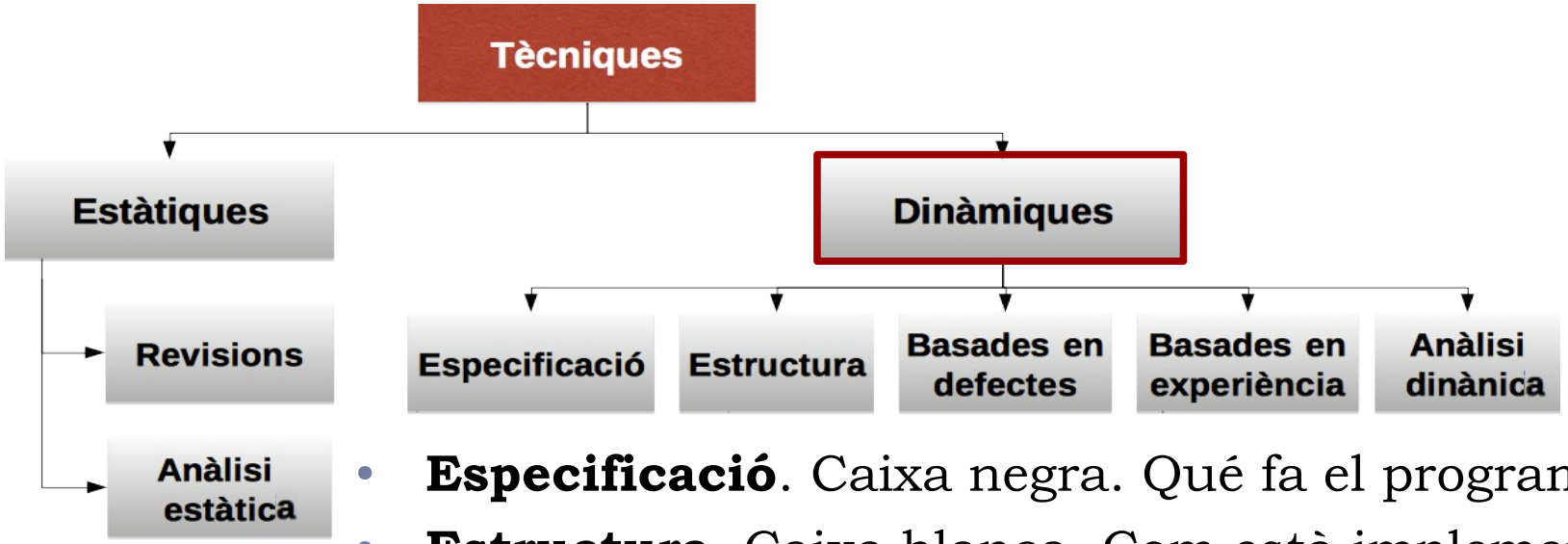
Verificació del programa

- **Tècniques de disseny de les proves**



Verificació del programa

- **Tècniques de disseny de les proves**

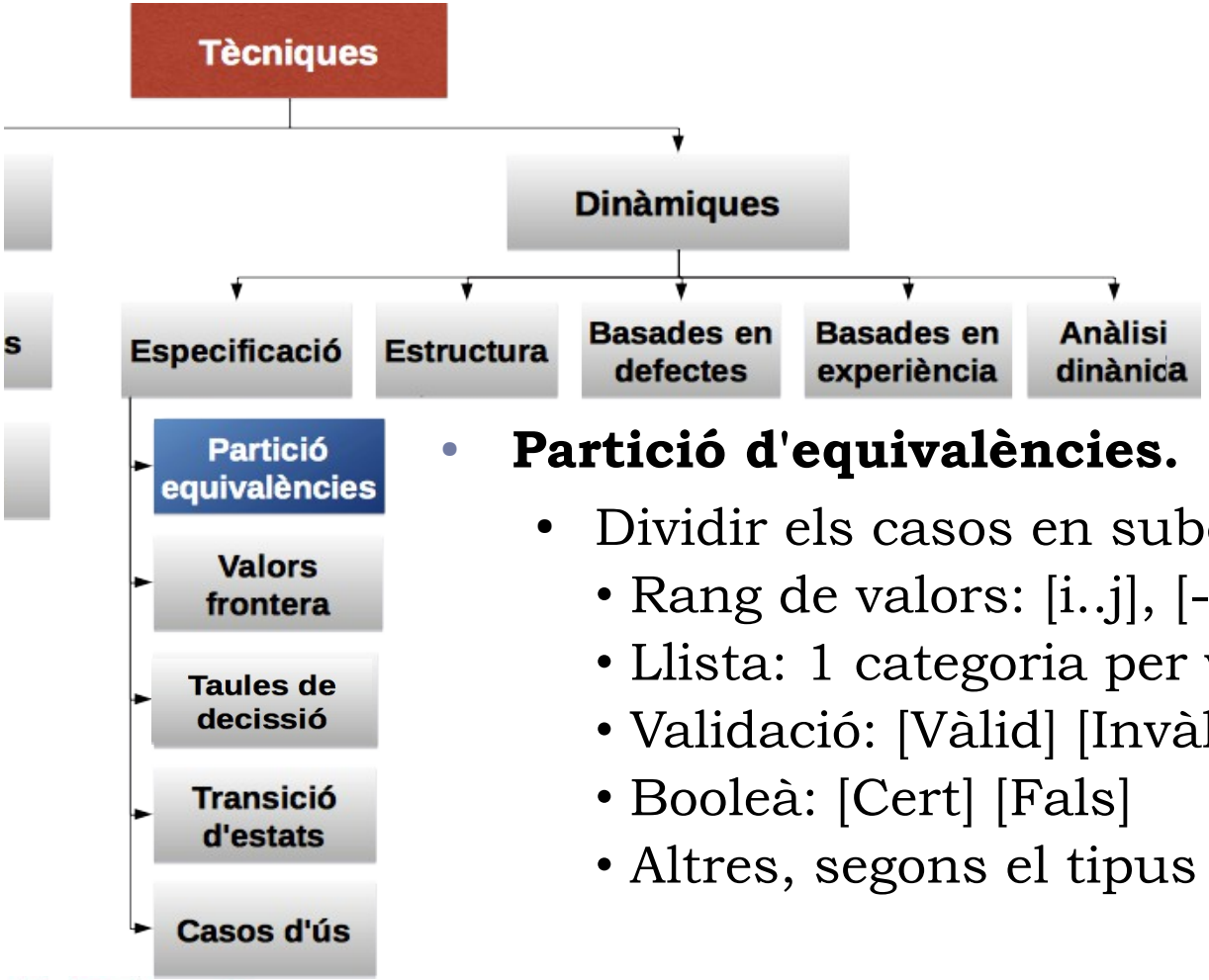


- **Especificació**. Caixa negra. Qué fa el programa.
- **Estructura**. Caixa blanca. Com està implementat el programa.
- **Basades en defectes**. Trobar categories de defectes.
- **Basades en l'experiència.**
- **Anàlisi dinàmica.** Anàlisi de l'aplicació en execució: memòria, xarxa...

21

Verificació del programa

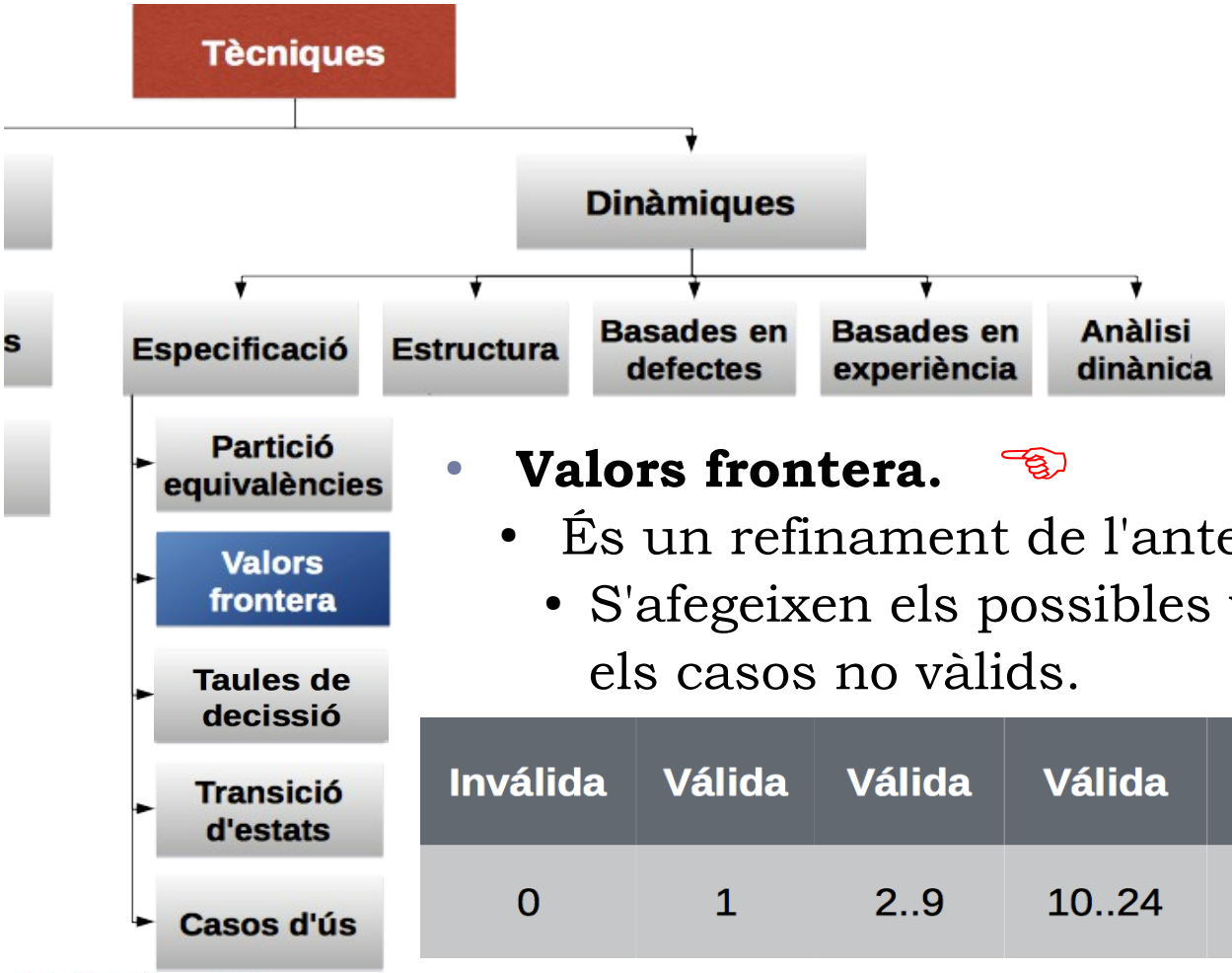
- **Tècniques de disseny de les proves**



- **Partició d'equivalències.**
  - Dividir els casos en subconjunts.
    - Rang de valors: [i..j], [-∞..i], [j..∞]
    - Llista: 1 categoria per valor
    - Validació: [Vàlid] [Invàlid]
    - Booleà: [Cert] [Fals]
    - Altres, segons el tipus de valor.

Verificació del programa

- **Tècniques de disseny de les proves**

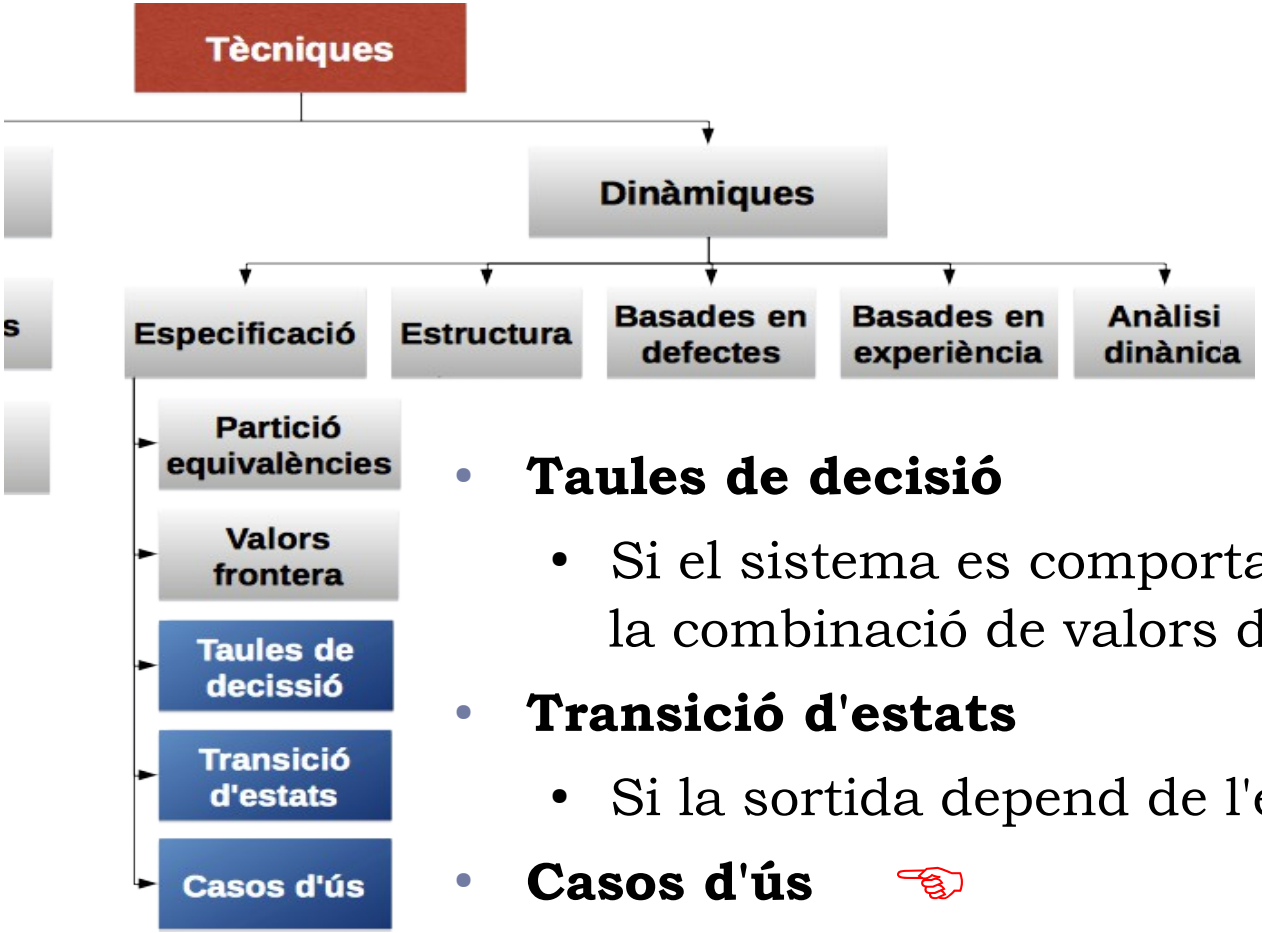


- **Valors frontera.** ☜
  - És un refinament de l'anterior.
    - S'afegeixen els possibles valors frontera i els casos no vàlids.

Inválida	Válida	Válida	Válida	Válida	Inválida
0	1	2..9	10..24	25..50	51

Verificació del programa

- **Tècniques de disseny de les proves**



- **Taules de decisió**
  - Si el sistema es comporta diferent segons la combinació de valors d'entrada.
- **Transició d'estats**
  - Si la sortida depend de l'estat del sistema.
- **Casos d'ús** ☞
  - Els casos d'ús es poden usar com base per a fer les proves.

Índex

- Objectius de la sessió
- Com es construeix un programa
- Implementació d'un programa:
  - Codificació de l'algorisme:
    - Edició del codi font en C.
  - Obtenir la versió executable i executar:
    - Procés de compilació.
    - Detecció i depuració d'errors.
    - Verificació del programa.
- **Exemple pràctic**
- Entorn de desenvolupament: MS-DOS i Eina Code::Blocks

Exemple pràctic

Enunciat de l'exercici:

Un amic nostre que viu en una casa adossada ens explica que aquest estiu el jardí li ha quedat destrossat i que s'està proposant arreglar-ho. Per a estalviar costos anirà a comprar el material i ho farà ell mateix.
Una zona important del jardí, amb àrea circular, la vol cobrir amb graves, de forma que el manteniment sigui mínim. S'ha mirat les possibles graves i inicialment n'ha escollit tres: rodó xocolata nº7 que val 159,00 euros el big-bag, rodó rosa nº5 que val 176,00 euros i la graveta rodona de riu que en val 27,30 euros.
Un big-bag és aproximadament 1 m^3. El nº de la grava indica la seva mida i en el cas d'una grava petita és suficient instal·lar un gruix de 5 cm però en una grava de mida gran és necessita contar un gruix superior.

El tema és que només es vol gastar 300,00 euros per al tema de la grava i vol avaluar quanta zona del jardí pot cobrir amb aquest pressupost segons el tipus de grava que utilitzi.

Exemple pràctic

Recordem les fases del **cicle de vida** del programari:

1. Identificar el problema a resoldre.
2. Analitzar les necessitats que ha de cobrir l'aplicació.
3. Dissenyar l'arquitectura de l'aplicació i el joc de proves
4. Implementar el programa
5. Instal·lar i mantenir l'aplicació

Exemple pràctic

Fases del cicle de vida:

1. Identificar el problema a resoldre.

Calcular el radi de jardí que es pot cobrir.

2. Analitzar les necessitats que ha de cobrir l'aplicació.

Què ens demanen resoldre?

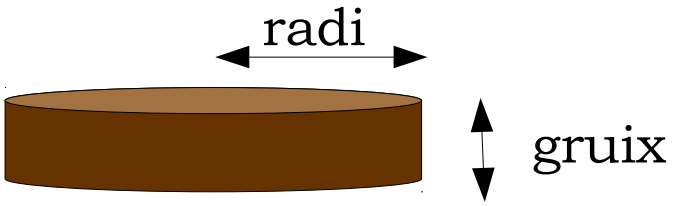
Exemple pràctic

2. Analitzar les necessitats que ha de cobrir l'aplicació.

El propietari vol gastar una certa quantitat de diners en grava. Té seleccionats tres tipus de material de diferents preus. Busquem els gruixos adequats per a cada grava:

Tipus grava	Preu m^3	Gruix recomanat
Rodó xocolata nº 7	159,00	10 cm
Rodó rosa nº 5	176,00	8 cm
Graveta rodona de riu	27,30	5 cm

Cal fer un programa que donat un tipus de grava (del que coneixem el preu m^3 i el gruix adequat) i el pressupost que es vol gastar calculi el volum que es vol cobrir.



Com la superfície és circular seria adequat que en lloc del volum es calculi el Radi del cercle.

Exemple pràctic

3. Dissenyar l'arquitectura de l'aplicació i el joc de proves

Donat un determinat tipus de grava es calcularà en funció del pressupost el radi del cercle de la superfície a cobrir i es mostrarà per pantalla.

COM resolem el problema (algorisme)

Fem una primera versió de l'algorisme indicant les accions que ha de fer el programa:

algorisme CalculRadi **es**

Inici

obtenir_dades necessàries
- obtenir_diners
- obtenir_preu_grava
- obtenir_gruix_grava
- obtenir_PI

- calcular_àrea_i_radi_cercle
- mostrar_radi

falgorisme

Volum grava = diners / preu grava(m^3)

àrea cercle= Volum grava/ gruix

$$radi = \sqrt{\frac{àrea}{\pi}}$$

Ens queda veure com refinem aquest algorisme detallant cada acció!

Exemple pràctic

3. Dissenyar l'arquitectura de l'aplicació i el joc de proves

Donat un determinat tipus de grava es calcularà en funció del pressupost el radi del cercle de la superfície a cobrir i es mostrarà per pantalla.

COM emmagatzemen la informació

Quina informació hem de guardar en el nostre cas?

diners
preu grava(m^3)
Volum grava
àrea cercle
gruix
radi

} Valors variables
Tipus real

π → Valor constant
Tipus real

Exemple pràctic

3. Dissenyar l'arquitectura de l'aplicació i el joc de proves

Refinant la versió anterior:

```
algorisme CalculRadi es
var       diners, preu, volum, area_cercle, gruix, radi : real;  fvar
constant  PI := 3.1416;  fconstant
inici
    escriure("Indica els diners que vols gastar (euros):");
    llegir(diners);                                          $ obtenir_diners
    escriure("Indica el preu de la grava (euros):");
    llegir(preu);                                            $ obtenir_preu_grava
    escriure("Indica el gruix de la grava (cm):");
    llegir(gruix);                                           $ obtenir_gruix_grava
    volum:= diners /preu;                                    $ calcular_radi_i_area_circle
    area_cercle := volum / gruix;
    radi:=arrel_quadrada(area_cercle/ PI);
    escriure("El radi de jardi que es pot cobrir es: ",radi);    $ mostrar_radi
falgorisme
```


Exemple pràctic

3. Dissenyar l'arquitectura de l'aplicació i el joc de proves

COM verifiquem que la solució és correcta

1. Hem de coneixer els rangs dels valors d'entrada:

   - Els rangs poden ser exactes o aproximats.
   - Amb les proves garantim que el programa funcionarà pels rangs indicats.

Variable	Rang
Preu m^3	[20,00 .. 180,00]
Gruix recomanat cm	[5 .. 10]
Dinners €	[10 .. 1.000.000]

Exemple pràctic

3. Dissenyar l'arquitectura de l'aplicació i el joc de proves

COM verifiquem que la solució és correcta

2. Dissenyem les proves (partició d'equivalència):

Prova	Entrades			Sortida
	Dinners (€)	**Preu grava (€/m^3)**	**Gruix grava (cm)**	**Radi (cm)**
Prova de fum	100	159,00	10	**1,4149**
Radi Màxim	1.000.000	20,00	5	**564,18957**
Radi mínim	10	180,00	10	**0,42052**

Exemple pràctic

4. Implementar el programa.

(i) Codificar l'algorisme utilitzant un llenguatge de programació:

```
#include <stdio.h>
#include <math.h>
```
llibreries i ctes

En C, el programa principal es denomina sempre **main**:
- Només pot haver un main.
- Es el primer que s'executa.

```
/* Programa Principal */
int main ()
{
    float diners, preu, gruix, volum, area, radi, gruix_m;

    printf( "\nIndica els diners que vols gastar (euros):   " );
    scanf( "%f", &diners );
    printf( "Indica el preu de la grava (euros):   ");
    scanf( "%f", &preu );
    printf( "Indica el gruix de la grava (cm):   ");
    scanf( "%f", &gruix );
    gruix_m = gruix / 100;                  /* Canvi d'unitats: cm → m */
    volum = diners / preu;
    area_cercle = volum / gruix_m;
    radi = sqrt(area_cercle / M_PI);        /* La constant PI esta definida a la llibreria math.h */
    printf("\nEl radi de jardi que es pot cobrir es: %.04f cm\n", radi);
    return 0;
}
```

var

inst.

4. Implementar el programa.

(ii) Obtenir la versió executable (Compilar i linkar). Executar i verificar que és la solució al problema.

- Executar les proves dissenyades i verificar que els resultats són correctes.

(iii) Documentar el procés.

- Documentació interna: correspon als comentaris que s'afegeixen al codi font i que l'ajuden a entendre. En C ho indicarem amb el caràcter situant el text entre: /* text */.
- Documentació externa: inclou dues de les fases del cicle de vida (anàlisi de l'aplicació i el disseny), el joc de proves i el manual d'usuari amb les instruccions per a executar el programa amb les explicacions del funcionament. També inclou el manual de l'administrador amb els requeriments per a què es pugui executar l'aplicació.

5. Instal·lar i mantenir l'aplicació

L'administrador del sistema serà l'encarregat d'instal·lar l'aplicació per a què els diferents usuaris la puguin utilitzar.

Els requeriments de l'aplicació poden variar al llarg del temps i si és el cas, aquesta tornarà al programador per a què s'actualitzi a les necessitats.

Índex

- Objectius de la sessió
- Com es construeix un programa
- Implementació d'un programa:
  - Codificació de l'algorisme:
    - Edició del codi font en C.
  - Obtenir la versió executable i executar:
    - Procés de compilació.
    - Detecció i depuració d'errors.
    - Verificació del programa.
- Exemple pràctic
- **Entorn de desenvolupament:** MS-DOS i Eina Code::Blocks

Entorn de desenvolupament MS-DOS

Línia de comandes (CMD)

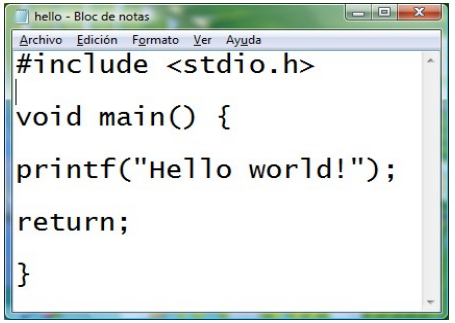
- És un mètode que permet als usuaris donar instruccions a algun programa informàtic per mitjà d'una línia de text simple.
- Es pot emprar interactivament, escrivint instruccions en alguna espècie d'entrada de text, o es pot utilitzar d'una manera molt més automatitzada (mode batch), llegint les instruccions des d'un arxiu (script).

Passos per compilar i executar un programa

1.- Modificar variable d'entorn PATH amb la ruta d'instal·lació del compilador mingw:

```
set PATH=C:\Archivos de programa\CodeBlocks\MinGW\bin;%PATH%
```

2.- Editar i guardar un arxiu C amb un editor de text p.e: notepad:



3.- Compilar i executar el programa:

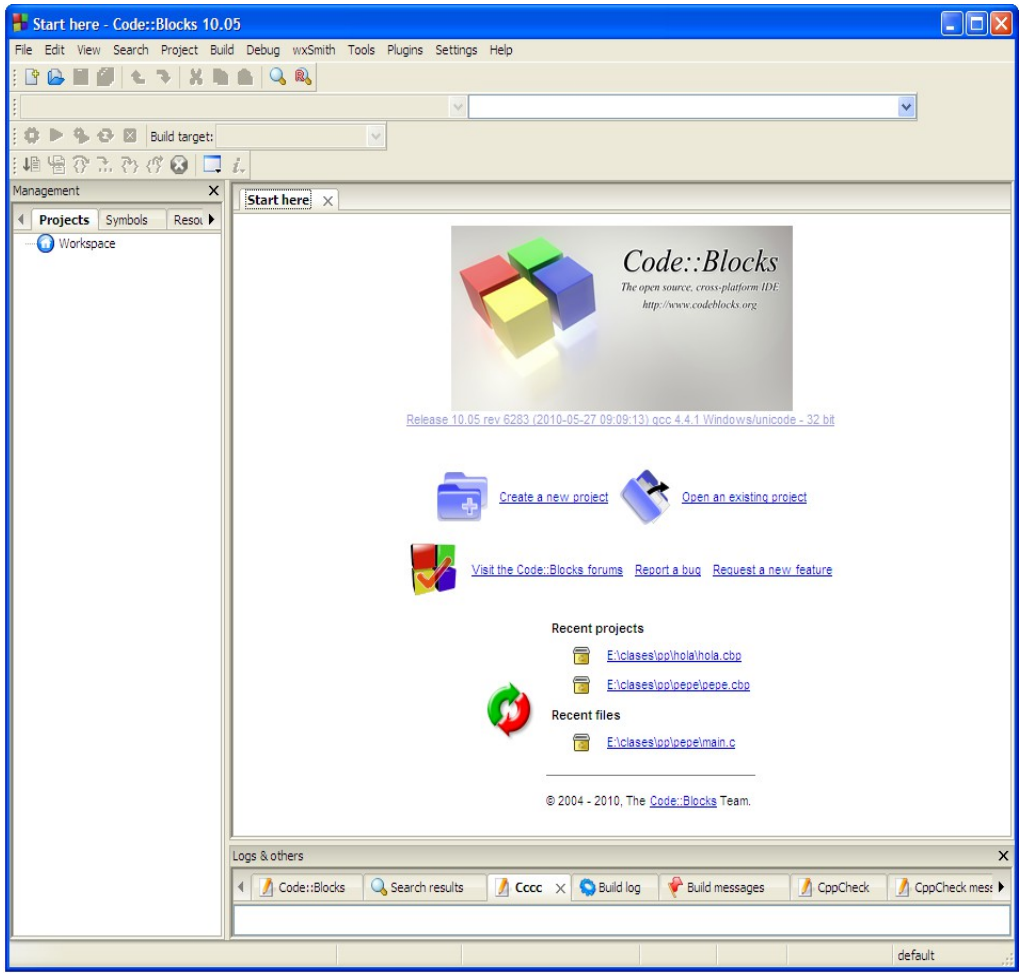
```
C:\Documents and Settings\Mis documentos>gcc hello.c -o hello.exe

C:\Documents and Settings\Mis documentos>hello
Hello world!
C:\Documents and Settings\Mis documentos>_
```

Entorn de desenvolupament Code::Blocks

- **Code::Blocks** és una eina **IDE** (entorn de desenvolupament integrat) que facilita eines per la creació i depuració de programes en C i C++. També inclou el compilador (MinGW) que s'instal·la conjuntament.
  - Permet gestionar un projecte amb múltiples fitxers
  - Facilita el manteniment del projecte: detecta els fitxers modificats
  - Dóna informació detallada dels errors i advertiments de compilació i enllaçat
  - Facilita la depuració dels programes i l'execució pas a pas.
- Distribució gratuïta.
- Per instal·lar Code::Blocks, teniu la documentació al moodle.

Entorn de desenvolupament Code::Blocks

Pantalla principal de Code::Blocks:



Part superior: menú i barra d'eines

Esquerra: Explorador de projectes i fitxers de codi

Centre: Àrea d'edició

Part inferior: Resultat de compilació (errors i advertiments).

Entorn de desenvolupament Code::Blocks

- Sempre heu de treballar amb **projectes**!! En el nostre cas, en llenguatge C:

  File->New->Project

  – Tipus de projecte:
    - Console application:
      Ens crea el fitxer principal de codi (main.c)
    - Empty project:
      Crea un projecte buit al qual podrem afegir fitxers després (Add files).

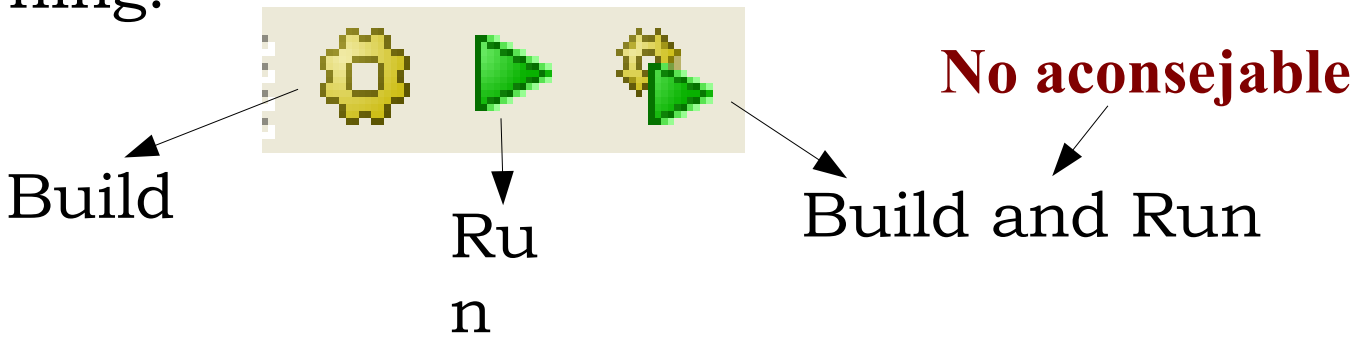
Entorn de desenvolupament Code::Blocks

- Hem de triar la carpeta `c:\Projectes\` com a base per als nostres projectes.
  Es crearà una carpeta, amb el nom del nostre projecte, amb tots els fitxers a dins: font, objecte, executable...

Entorn de desenvolupament Code::Blocks

- **Compilar**. Abans de compilar primer cal configurar correctament el compilador:
  Settings->Compiler

- **Executar**. Abans d'executar un programa primer cal verificar que no contè cap error ni warning.



Tasques a realitzar al laboratori:

A partir de l'exemple que us proporcionem:

1. Obteniu la versió executable de dues maneres:
   - fent servir l'entorn MS-DOS.
   - fent servir l'entorn Code::Blocks.

2. Executeu el programa:
   - fent servir l'entorn MS-DOS.
   - fent servir l'entorn Code::Blocks.

3. Amb el joc de proves verifiqueu el correcte funcionament del programa.