1. Pseudocodi, codificació en C i eines per a la programació

1. Què és programar?

La programació és un procés en què es genera una seqüència d'instruccions que un ordinador pot interpretar i executar, i que serveixen perquè aquest ordinador resolgui de manera automàtica un problema concret.

A continuació, teniu un exemple d'un programa C que escriu en pantalla la frase «Hello, World!».

```
#include <stdio.h>
```

```c
int main() {
   printf("Hello, World!");
   return 0;
}
```

Com veieu, un programa no és més que un text en què hi ha una sèrie de paraules clau (*int*, *return*) que l'ordinador entén i que pot processar per donar un resultat. El text es mostra en colors per ajudar-nos a identificar d'un cop d'ull les paraules clau, però és només una ajuda visual, ja que un programa està format únicament per text, res més.

En aquesta assignatura, aprendrem a generar programes que ens permetin utilitzar l'ordinador perquè faci les tasques que nosaltres vulguem.

El procés de programar segueix uns passos molt importants:

1. Dissenyar un algorisme que solucioni el problema.
2. Codificar l'algorisme perquè l'entengui l'ordinador.
3. Provar el programa.

La programació sempre hauria de contemplar aquests tres passos, que detallarem en els apartats següents.

1.1. Què és un algorisme i com es genera? El pensament computacional

Abans de crear un programa, cal reflexionar, tenir molt clar quin problema ha de resoldre, els casos que ha de cobrir i assegurar-se que els resultats que proporcionarà seran correctes. Per a això, es dissenya un algorisme.

Un **algorisme** és un conjunt de passos ordenats que cal seguir per executar una tasca o resoldre un problema.

Un algorisme no és quelcom exclusiu de la programació; de fet, utilitzem algorismes contínuament en la vida quotidiana. Per exemple, quan cuinem seguint una recepta d'un llibre de cuina. Una recepta és una sèrie de passos (tallar i pelar verdures, afegir diferents ingredients a una paella en un ordre determinat, etc.) que ens permeten arribar a la solució del problema: fer una truita de patates o una pizza.

Per tant, dissenyar un algorisme no és programar, però per programar sí que cal generar

prèviament un algorisme. Un algorisme no és una solució única a un problema, sinó que hi ha moltes solucions que es poden adoptar per a un mateix problema. Però, perquè un programa sigui robust i no generi errors, és important que estigui ben dissenyat. És a dir:

- Que es prevegin totes les possibles situacions que es poden presentar (per exemple, si diem que per fer una recepta necessitem dues patates, què passa si són grans?, i si són petites?). Cal plantejar-se sempre totes les possibles opcions.
- Que sigui òptim, és a dir, que el resultat sigui l'esperat, minimitzant els recursos.
- Que estigui ben estructurat perquè sigui fàcil entendre què fa.

Per generar un algorisme que prevegi totes aquestes característiques, convé aplicar el que es coneix com a **pensament computacional**. El pensament computacional ajuda a generar algorismes òptims i robustos. Com hem dit, els algorismes no són exclusius de la programació, per tant, el pensament computacional no només s'aplica a la programació, sinó que és una manera de pensar que fem servir en moltes situacions. Així, doncs, la programació és el pensament computacional aplicat als ordinadors.

El pensament computacional estableix els passos següents per resoldre un problema:

1. **Descompondre el problema** que volem solucionar en parts que siguin més petites i fàcils de resoldre o tractar.
   En el cas de la recepta, les diferents parts podrien ser preparar els ingredients (pelar i tallar verdures, batre els ous, escalfar l'oli), cuinar-los (fregir les verdures, afegir els ous al foc, condimentar amb sal i pebre o amb altres espècies, etc.) i servir-los.
2. **Reconèixer els patrons** que es manifesten en el problema i que ens permeten repetir el mateix procediment diverses vegades. Per exemple, si cal pelar i tallar quatre patates, tres tomàquets i dues pastanagues, repetirem el mateix patró de pelar i tallar diverses vegades.
3. **Abstreure el problema** per enfocar-nos únicament en la informació rellevant per al problema concret. En el cas de la recepta, serà important saber si els ingredients estan en bon estat, però en el moment de cuinar-los no ens importarà saber quin preu tenien a la botiga o si són orgànics o no. Això serà decisió del xef, però no és rellevant per executar la recepta.
4. **Generar l'algorisme** amb la solució pas a pas per resoldre el problema.

Al llarg de l'assignatura practicarem aquest exercici sobre el pensament computacional. Si us interessa aprendre'n més, podeu consultar aquest **enllaç (https://www.bbc.co.uk/bitesize/topics/z7tp34j)** .

1.2. Exemple de generació d'un algorisme

Tornem a l'exemple de la recepta de la truita. Si seguim els quatre passos del pensament computacional que hem descrit abans, ens adonarem del següent:

- **Descompondre el problema**. L'elaboració d'una truita consisteix a preparar els ingredients (pelar i tallar les patates, trencar i batre els ous), cuinar-los (coure les patates, afegir-hi els ous, etc.) i servir-los.
- **Reconèixer els patrons**. Cadascuna de les patates es processa de la mateixa manera, igual que cadascun dels ous.
- **Abstreure el problema**. Per cada patata que usem, necessitarem un ou i mig, més o menys. Les quantitats dels ingredients són essencials per elaborar una recepta i haurem de treballar amb les quantitats. En canvi, no ens importarà si els ous són rossos o blancs. Per tant, el nostre algorisme haurà de tenir en compte les quantitats dels ingredients, però no el color dels ous.
- **Generar l'algorisme**. Amb la informació anterior podrem generar l'algorisme següent:

  Ingredients i estris (variables):

  - Bol
  - Paella
  - Fogó
  - Patates
  - 4 ous
  - Oli
  - Sal
  - Ceba

  Passos que cal seguir (algorisme):

  1. Trencar els ous i posar-los al bol.
  2. Batre els ous al bol fins que quedin uniformes.
  3. Pelar les patates.
  4. Tallar les patates en trossos no gaire grans.
  5. Si posem ceba, pelar-la.
  6. Si posem ceba, tallar-la en trossos no gaire grans.
  7. Engegar el foc.
  8. Posar la paella al foc.
  9. Posar l'oli a la paella.
  10. Quan l'oli estigui calent, fregir les patates.
  11. Si posem ceba, fregir-la.

12. Posar els ous i les patates en un bol.
13. Si posem ceba, afegir-la al bol.
14. Barrejar el contingut del bol.
15. Posar el contingut del bol a la paella.
16. Esperar fins que adquireixi una forma consistent.
17. Girar la truita.
18. Quan la truita estigui feta, apagar el foc i posar-la al plat.

2. Disseny d'algorismes: pseudocodi i diagrames de flux

En el cas de la programació, el procés de generar un algorisme és molt semblant a l'exemple de la truita, però el llenguatge que s'utilitza sol ser més pròxim als diferents llenguatges de programació que hi ha.

Els llenguatges de programació, com C, Java, Python o d'altres, tenen una «sintaxi» molt estricta. Encara que intenten assemblar-se al llenguatge natural, és a dir, el que utilitzem les persones quan ens comuniquem, hi ha una sèrie de regles que cal seguir, i és imprescindible que les línies de codi estiguin escrites d'una manera determinada. En aquesta assignatura, anirem aprenent aquestes regles, a poc a poc i amb pràctica.

Cada llenguatge de programació té la seva sintaxi específica, que s'aprèn amb la pràctica. Però l'algorisme que utilitzem per solucionar un problema pot ser igual o molt semblant, independentment del llenguatge de programació que utilitzem per implementar-lo. Per això, en la fase de disseny de l'algorisme, s'utilitza el que s'anomena **pseudocodi** o **diagrama de flux**.

2.1. Pseudocodi

El pseudocodi és una descripció pas a pas de l'algorisme que s'ha dissenyat.

Es diu que el pseudocodi és una descripció a molt alt nivell de l'estructura d'un programa. Amb *alt nivell* volem dir que les instruccions o els passos que s'utilitzen són fàcils d'entendre per a la persona que l'està generant (o qualsevol altra persona que el llegeixi). Encara que s'utilitzin estructures típiques de la programació, s'empra un llenguatge que s'entén de manera més o menys

immediata.

D'altra banda, hi ha altres llenguatges de programació que es diuen de *baix nivell* en què les instruccions són més pròximes al que el processador ha de fer per generar la solució al problema.

El pseudocodi no té en compte la sintaxi del llenguatge de programació que esmentàvem abans, sinó que és molt més flexible.

A continuació, us posem com a exemple el pseudocodi per al codi de programa «Hello World!» que hem vist abans. Podeu veure les diferències entre tots dos?

Pseudocodi	C

algorithm HelloWorld
 	
writeString("Hello World!");
 	
end algorithm
	`#include <stdio.h>`
	
	`int main() {`
	`  printf("Hello, World!");`
	`  return 0;`
	`}`

A mesura que avanci l'assignatura, anirem aprenent a poc a poc com s'escriu el pseudocodi. Encara que és més flexible que la sintaxi d'un llenguatge de programació, sí que establirem algunes regles que ens ajudaran al fet que, en generar l'algorisme, ja tinguem al cap la implementació en un llenguatge de programació (en el nostre cas, C). En la **PAC 0 - Nomenclàtor de pseudocodi (https://aula.uoc.edu/courses/78741/pages/rec-nomenclator-de-pseudocodi-2-2)** , us presentem un resum de les regles que heu de seguir a l'hora de generar un algorisme usant pseudocodi. Tingueu-les a mà, perquè segur que l'haureu d'utilitzar unes quantes vegades en aquest curs.

Recordeu

Un algorisme en pseudocodi no és un programa. És la descripció de la solució que s'ha dissenyat per resoldre el problema plantejat.

Perquè aquesta solució es converteixi en un programa, cal codificar-la en el llenguatge de programació triat (en el nostre cas, C).

2.2. Diagrames de flux

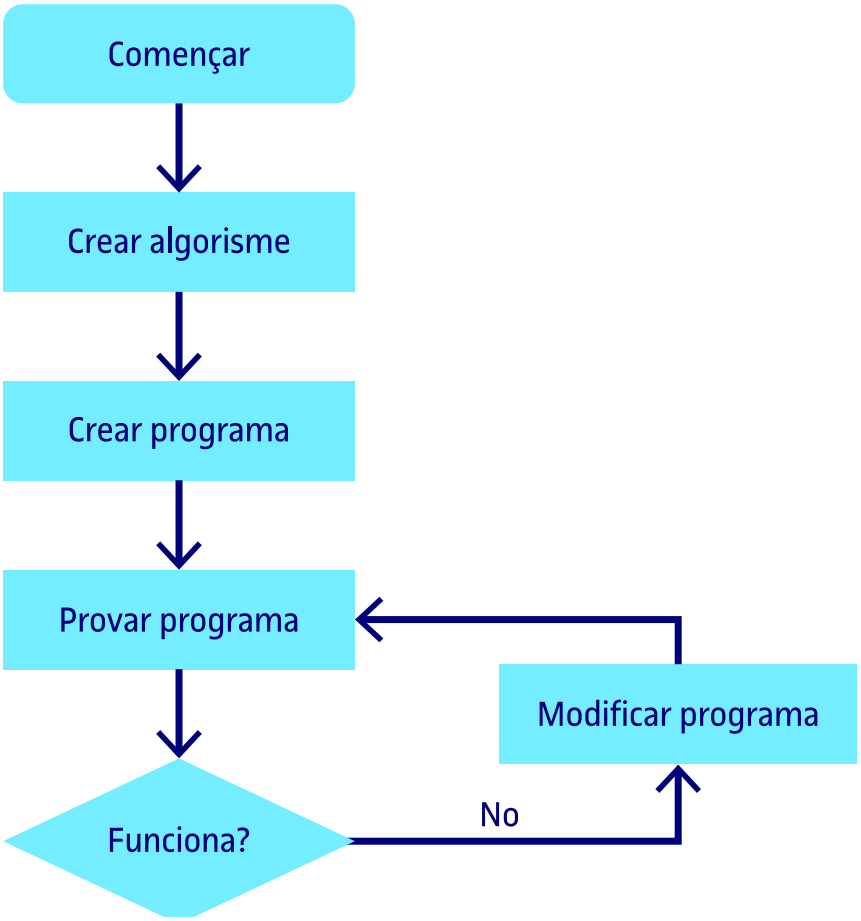
Com us hem comentat anteriorment, el pseudocodi és molt útil a l'hora de conceptualitzar l'algorisme que soluciona el problema. A més, ens ajuda a plantejar-lo d'una manera que serà molt similar al programa final i, per tant, ens facilita la feina.

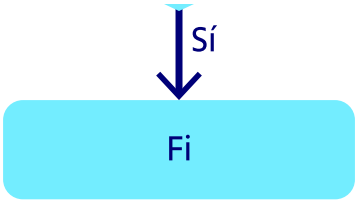
Així i tot, hi ha vegades en què una eina més visual també pot ser útil. És el cas dels diagrames de flux.

Els **diagrames de flux** són esquemes en què es poden distribuir les instruccions que formen un algorisme, de manera que visualment es pot tenir una idea de l'estructura, cosa que facilita la comprensió de les instruccions descrites.

Un diagrama de flux pot ser molt útil com a complement del pseudocodi. Un algorisme descrit en un diagrama de flux és més fàcil d'interpretar d'un cop d'ull. Així i tot, sempre cal descriure'l usant pseudocodi, perquè ens acosta a la forma final que tindrà el programa, ja que l'estructura de les instruccions —que s'executen una darrere de l'altra— de l'algorisme en pseudocodi és molt semblant a la d'un codi en un llenguatge de programació.

Tot seguit, us mostrem un diagrama de flux que descriu el procés que cal seguir per generar un programa informàtic:





3. Codificació

La **codificació** consisteix a traduir l'algorisme que hem dissenyat per resoldre un problema —sigui amb un diagrama de flux o en pseudocodi— i transformar-lo en un programa escrit en un llenguatge concret de programació (en el nostre cas, C).

La diferència principal entre un algorisme escrit en principal i un programa codificat en C és que l'algorisme és un disseny dels passos per resoldre un problema, però no es pot executar. En canvi, el programa codificat en llenguatge C sí que pot ser executat per un ordinador, i ens dona els resultats que estem buscant per resoldre el problema.

Així doncs, per escriure un programa (en el nostre cas, en llenguatge C) únicament podrem utilitzar les seves instruccions i normes bàsiques perquè el codi pugui ser compilat i executat sense problemes. Per tant, haurem d'adaptar l'algorisme al llenguatge C.

Quan escrivim un algorisme en pseudocodi, podem utilitzar un llenguatge que ens sigui més clar per entendre com serà el desenvolupament del programa. Encara que seguirem una sèrie de regles per generar algorismes en pseudocodi, un dels avantatges d'aquest és que el llenguatge és més natural. De fet, encara que en aquesta assignatura escriurem els algorismes en anglès, es podrien escriure perfectament en català. En canvi, si escrivim en llenguatge C (o qualsevol altre llenguatge de programació), haurem d'observar estrictament totes les regles de la sintaxi del llenguatge en qüestió, les paraules clau o reservades, etc. Si ens oblidem d'alguna de les regles, cometrem errors i el programa no es podrà executar.

3.1. Programa en C

En aquest punt, farem una anàlisi de l'estructura d'un programa en C.

Reprenem l'exemple del programa «Hello World!» que hem mostrat abans.

```c
#include <stdio.h>

int main() {
   printf("Hello, World!");
   return 0;
}
```

Un programa en C sempre ha de tenir una funció `main()`. Ja hem introduït aquesta funció en **REC. Preguntes freqüents (https://aula.uoc.edu/courses/78741/pages/rec-preguntes-frequents-2-2)**, però ara la descriurem amb més detall. Aquesta és la funció principal del programa. Més endavant, veurem exactament el que és una funció, però en aquest moment considerarem una funció com una porció de codi que fa unes operacions determinades i retorna un valor com a resultat.

La funció es defineix de la manera següent:

```c
int main() {}
```

Compte!

Per generar programes utilitzarem **CodeLite**. Tractarem els detalls d'aquesta eina durant el curs, però, en aquest punt, només heu de saber que CodeLite és un programa que s'usa per crear-ne d'altres, de la mateixa manera que Word i Google Docs s'utilitzen per generar documents de text, o Excel i Google Sheets s'usen per crear fulls de càlcul.

Quan CodeLite crea un programa, veureu que la funció `main` té més dades entre els parèntesis, i s'assembla al següent:

```c
int main(int argc, char **argv)
```

Els anomenats «paràmetres» de la funció `main` s'afegeixen perquè, un cop creat el programa (el fitxer executable), es pugui anomenar amb paràmetres per canviar-ne la funcionalitat.

No patiu si no ho enteneu en aquest moment. Podeu deixar els paràmetres o esborrar-los, però simplement ignoreu aquesta diferència entre el que us expliquem i el que apareix en CodeLite.

A continuació, analitzem la línia detalladament:

- `int`: la funció `main` ha de retornar un valor. En aquest cas, `int` (*integer*) informa que aquest valor és un nombre enter (en les unitats següents veurem amb més detall el que significa això). Normalment, el valor de retorn de la funció `main` estarà associat amb les incidències que puguin ocórrer en el programa. En altres funcions, el valor de retorn és el resultat d'un processament de dades. Ara com ara, no hi donarem gaire importància.
- `main()`: aquesta és la funció `main`. Els parèntesis s'utilitzen per determinar els paràmetres amb què es diu la funció. En aquest cas, la funció `main` no té paràmetres.
- `{}`: les línies del programa se situen entre aquestes claus.

En aquest cas, el programa el formen dues línies de codi:

- `printf("Hello, World!");`
  Aquesta línia és la que fa que es mostri «Hola, món!» en la pantalla.
- `return 0;`
  Com hem comentat, la funció `main` retorna un nombre enter. En aquest cas, simplement retorna 0. En aquest punt, no cal que entengueu per què ho fem així.

Fixeu-vos que abans de la funció `main` tenim una altra línia, que comença amb #. En C, les línies que comencen amb # són les que es diuen **preprocessadores**. Aquestes línies tenen diferents objectius. En aquest cas concret, la línia `#include <stdio.h>` serveix perquè el programa carregui la llibreria estàndard de C, que es diu `<stdio.h>` (*standard input/output*). Les **llibreries estàndard de C** són conjunts de funcionalitats que estenen les instruccions bàsiques del llenguatge C, de manera que faciliten fer certes tasques. La llibreria `stdio.h` conté la informació necessària per poder utilitzar la funció `printf`, que s'empra en aquest programa.

3.2. Codificació, sintaxi, compilació i execució

Probablement us deveu estar preguntant si, quan un programa és un fitxer de text i teniu molts fitxers de text a l'ordinador, podríeu intentar executar un d'aquests fitxers d'alguna manera.

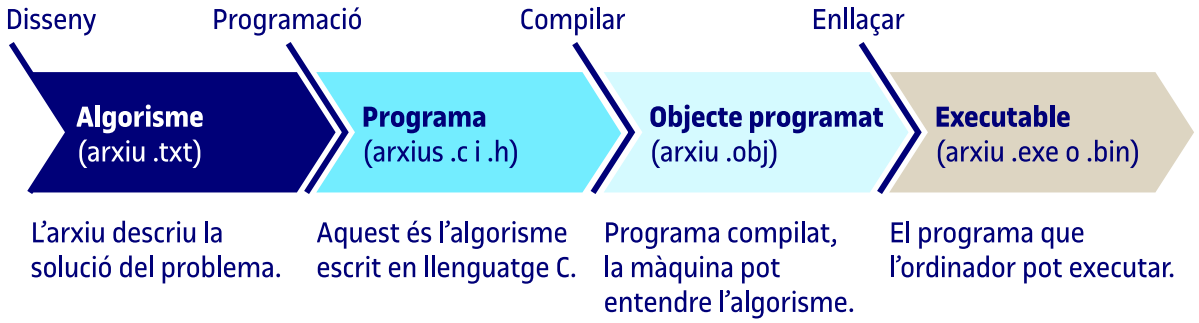
La resposta és que podríeu intentar-ho, però heu de seguir dos passos en el cas del llenguatge de programació C: compilar-lo i enllaçar-lo.

- **Compilar (***compile***)**: el compilador o *compiler* analitza el codi que hem escrit i si hem seguit les regles del llenguatge de programació (en el nostre cas, C). Aquestes regles són el que es diu *sintaxi*. Si hi ha alguna regla de C que no s'ha seguit, el compilador ens informarà que hi ha un error. En canvi, si hem escrit un programa correcte segons les regles de la sintaxi de C, la

compilació serà correcta i es generarà un fitxer .obj per a cada fitxer de codi (.c) que es compili. Aquest fitxer .obj (objecte de codi o *code object*) és una traducció del llenguatge de programació a un llenguatge que el processador de l'ordinador pot entendre. Tingueu en compte que el resultat de compilar encara no és un programa.

- **Enllaçar (*link*)**: és el procés de crear un únic fitxer executable a partir dels fitxers .obj que hem compilat anteriorment. L'enllaçador o *linker* utilitzarà tots els fitxers .obj que necessiti per generar el programa. Si hi ha algun fitxer .obj que no es troba o alguna incoherència entre els fitxers .obj (per exemple, el mateix codi és a dos llocs diferents o, a l'inrevés, aquest codi no és a cap dels fitxers), l'enllaçador generarà un error. Si, al contrari, tot és correcte, l'enllaçador generarà un programa, i aquesta vegada sí que el podrem executar. Els fitxers executables poden tenir diverses extensions, però normalment són .exe (en Windows) o .bin (en Linux).

Com que sempre s'enllaça després de compilar, CodeLite ens dona l'opció de construir: compilar i enllaçar en una única acció.



- **Executar**: el fet que el programa compili i enllaci (*compiles* i *links*) sense errors no vol dir que funcioni correctament. Només indica que no hi ha errors sintàctics en el programa i que es pot executar. Una vegada creat el programa, cal provar-lo o testar-lo. El primer test consisteix a executar-lo i assegurar-se, ràpidament, que el programa funciona i fa el que necessitem que faci.
- **Testar**: un cop que sabem que el programa es pot executar, és molt important que comprovem que els resultats són correctes. Per això, cal que ens assegurem que tots els casos possibles estan previstos en el funcionament del programa. Per exemple, si creem un programa que suma dos nombres enters, hauríem de provar que funciona en diferents casos:

    - quan els dos nombres són positius,
    - quan els dos nombres són negatius,
    - quan el primer és negatiu i el segon, positiu, i
    - quan el primer és positiu i el segon, negatiu.

    És molt important fer aquest test perquè, si no es fa, pot ser que fem que el programa ens doni resultats equivocats o, fins i tot, que es «pengi» l'ordinador. Com podeu comprovar en el cas

anterior, fins i tot una cosa tan fàcil com sumar dos nombres requereix que es provi diferents vegades. Us semblen suficients les opcions que hem plantejat anteriorment? Creieu que caldria provar també els casos en què un dels dos nombres sigui zero? Segurament, sí. És més complicat del que sembla, veritat?

Testar un programa és una tasca molt important i, de vegades, per fer-ho es generen el que es diuen **jocs de prova**. Aquests jocs de prova són diferents casos d'entrada, cadascun dels quals dona un resultat de sortida. Es dissenyen tants casos d'entrada com situacions diferents trobem, i per a cada conjunt de valors d'entrada es desen els valors de sortida. Aquests jocs de prova són molt útils per assegurar que el programa funciona correctament, especialment en el cas en què s'hagi fet algun canvi en el programa. Si és així, podem usar automàticament el mateix joc de proves i veure si els valors de sortida són els mateixos que abans de la modificació.

Recordeu

Superar un joc de proves és una manera d'assegurar-se que el programa és més robust que si no el superés, però no hi ha cap joc de proves prou extensiu per permetre assegurar que el programa funcionarà sempre. Això no vol dir que no hàgim d'intentar fer el programa el més robust possible.

En aquesta assignatura, aprendrem estratègies per fer-ho, i també utilitzarem eines externes (en concret, DSLab) que ens ajudaran a testar que el programa funciona correctament.

Resum

- La **programació** és un procés en què es genera un **programa**, una seqüència d'instruccions que un ordinador pot interpretar i executar, i que serveixen perquè aquest ordinador resolgui de manera automàtica un problema concret.
- En programar, generem un **algorisme**, que és un conjunt d'instruccions que serveixen per resoldre un problema d'una manera general. No tots els algorismes són programes, però tots els programes contenen un algorisme.
- Els algorismes se solen escriure en **pseudocodi**, que l'ordinador no entén. Els programes s'escriuen en **codi** en un llenguatge de programació (en el nostre cas, C), que l'ordinador sí que

pot entendre.

- El **pensament computacional** és un mètode que s'aplica per generar algorismes. Consisteix a seguir aquests passos:
  - **Descompondre el problema**.
  - **Reconèixer els patrons**.
  - **Abstreure el problema**.
  - **Generar l'algorisme**.
- Una vegada que tenim un algorisme, el codifiquem per fer un programa. En aquesta assignatura, per codificar el programa utilitzarem l'IDE **CodeLite**.
- Un cop creat el programa, cal **compilar-lo** i **enllaçar-lo** perquè el processador de l'ordinador el pugui entendre.
- Quan tinguem el programa finalitzat, caldrà testar-lo i assegurar-se que funciona com esperem.