Memòria del treball

L'ordinador com a eina de càlcul i simulació a l'ensenyament de les Ciències Experimentals i les Matemàtiques

Autor: Francisco Javier Ruiz Vegas
Curs: 2000-2001

Índex

Relació de material que acompanya a la memòria

1. Introducció

Són molts els estudis que s'han dedicat a estudiar el paper dels ordinadors a l'ensenyament en general. La utilització d'aquests com a eina de consulta, de comunicació i de plataforma on resoldre tot tipus d'exercicis s'ha generalitzat en els últims anys. Cada vegada, el professorat disposa de més programes que aprofiten els recursos multimèdia per fer més atractius els aspectes repetitius del procés d'aprenentatge.

Malgrat tot, el professorat de les disciplines científiques, i sobretot, aquells i aquelles que seguim amb interès el rumb que pren la Ciència actual, trobem a faltar, a més de tots els usos que actualment es fan de l'ordinador, un ús més "científic", amb concordança amb el vertader potencial que té l'ordinador per fer càlculs i simulació.

En molts camps de la Ciència i l'Enginyeria cada vegada és més freqüent la utilització de l'ordinador com una eina de treball, i aquest fet s'ha de tenir en compte a l'hora de formar els futurs científics i tècnics. Per això, és important que els estudiants dels ensenyaments secundaris, sobretot de batxillerat, coneguin els seus avantatges.

L'aparició dels ordinadors personals va obrir un nou món de possibilitats per a l'estudi de les matèries pròpies d'un estudiant de Ciències. Ara bé, quan examinem els continguts dels llibres de text observem que l'impacte d'aquesta revolució és mínim, perdent d'aquesta forma la possibilitat d'analitzar problemes més reals i interessants.

Per tant**, l'habilitat per fer servir els ordinadors és, en aquests moments, una habilitat imprescindible en la investigació científica i és també imprescindible que aquesta habilitat sigui adquirida tan aviat com sigui possible per poder crear estructures mentals sòlides com l'escriptura o el càlcul.**

Precisament, algunes de les característiques de l'actual sistema educatiu –com ara els crèdits variables, els itineraris curriculars individualitzats, les matèries optatives, el treball de recerca de batxillerat o la major autonomia dels centres per adaptar els objectius i continguts de les matèries– afavoreixen la introducció de l'ordinador com a eina de l'ensenyament en la major part de les disciplines.

1.1 Els ordinadors a l'Ensenyament

Avui dia, dins el món educatiu, es considera l'ordinador com un recurs de comunicació que ajuda a aprendre amb continguts multimèdia i que permet connectar-nos amb altres ordinadors de tot el món.

A l'Educació Infantil, Primària i Secundària, el tipus de programes i aplicacions informàtiques estan consolidant-se en tres àmbits:

a) **Aplicacions generals**: processadors de textos, bases de dades, fulls de càlcul, programes de dibuix, navegadors d'INTERNET...

b) **Aplicacions curriculars específiques**:

- Programes tancats d'ensenyaments assistits per ordinador (EAO) clàssics destinats a l'exercitació i la memorització.

- Programes d'informació i consulta: enciclopèdies en CD ROM, tutorials amb hipertextos, etc.

- Programes més oberts i interactius: simulacions interactives, algunes aplicacions creades pel propi professorat, etc.

c) **Entorns de creació d'aplicacions educatives**, representades per programes com CLIC i els llenguatges d'autor.

Els centres educatius pioners van començar a ensenyar Informàtica amb l'estudi de la Programació, dominada primerament pel BASIC, després pel PASCAL i el LOGO. Després es va substituir aquestes disciplines per l'aprenentatge d'aplicacions estàndards: tractament de textos, fulls de càlcul, bases de dades, programes de dibuix... També es van fer servir a l'aula aplicacions educatives específiques creades per alguns professors i distribuïdes pel Programa d'Informàtica Educativa (PIE) en les quals l'alumne és un usuari que pot canviar només alguns dels paràmetres d'aquesta aplicació.

L'evolució dels ordinadors, quant a rapidesa i velocitat, va permetre als centres introduir-se en el món multimèdia. La consulta d'enciclopèdies en CD-ROM va convertir-se en un dels usos freqüents de les aules d'informàtica.

Per últim, la introducció d'INTERNET als centres educatius ha tornat a canviar els continguts de les activitats realitzades a l'aula d'ordinadors. Al principi, només el professor de matemàtiques (gairebé sempre era d'aquesta matèria), entrava a l'aula d'informàtica. Ara, per sort, les aules han deixat de ser aules d'informàtica per convertir-se en aules d'ordinadors, i són pràcticament totes les disciplines les que fan servir els ordinadors com a suport de la matèria específica, incloent-hi les àrees de llengües, que han passat de tenir pràcticament vedada l'entrada a les aules d'ordinadors a convertir-se en els principals usuaris.

De totes formes, encara que és clar que la principal finalitat de l'ordinador és la comunicativa, s'ha deixat una mica per banda l'ús de l'ordinador com a eina de càlcul i de simulació, que són dos dels usos més freqüents en la ciència actual. Per a què l'alumne pugui treure el major profit de l'ordinador com a eina de càlcul i simulació no n'hi ha prou amb proporcionar-li programes en els quals aquest faci només d'usuari sinó que traurà un més alt rendiment en la comprensió de la matèria en qüestió si és ell mateix qui construeix l'aplicació.

1.2 La importància dels ordinadors en la Ciència actual

Podem dividir l'ús dels ordinadors en la Ciència en les següents categories:

- Càlcul i anàlisi numèric.

- Càlcul simbòlic.

- Simulació.

- Recopilació i anàlisi de dades.

1.2.1 Càlcul i anàlisi numèric

En molts dels problemes que sorgeixen en diferents branques de la Ciència es requereix resoldre equacions o sistemes d'equacions (algebraiques, transcendents o diferencials) que són pràcticament irresolubles, ja sigui perquè no hi hagi un mètode analític per trobar la solució o bé perquè aquest sigui molt llarg o molt difícil d'aplicar[1]. És aquí quan l'ordinador, fent servir tècniques d'anàlisi numèric, permet obviar la dificultat d'aquests càlculs i trobar solucions amb la precisió requerida.

¿Per què el càlcul numèric s'ha convertit en una part tant important en Ciències? Una de les raons és que les eines matemàtiques analítiques com el càlcul diferencial serveixen per resoldre els anomenats *problemes lineals,* és a dir, problemes en els quals un petit canvi de les condicions inicials porta només a un petit canvi de les solucions. No obstant, molts fenòmens naturals són *no lineals* (de fet, la majoria), i un petit canvi de les condicions inicials pot produir un gran canvi de les solucions. Degut a que només un petit nombre de fenòmens no lineals poden ser resolts amb mètodes analítics, l'ordinador es converteix en l'eina principal per explorar els fenòmens no lineals. Una altra raó per justificar la importància dels ordinadors és el creixent interès per sistemes amb moltes variables o amb molts graus de llibertat.

1.2.2 Càlcul simbòlic

Són molts els programes de càlcul simbòlic que hi ha ara al mercat: "*Mathematica*, *DERIVE*, *MATLAB*, *MAPLE* …" Aquests programes permeten manipular expressions simbòliques de forma que, plantejaments abans impensables, per la seva complicació i limitació de mitjans, són avui possibles. Aquests tipus de programes ens han de fer canviar l'estratègia de l'aprenentatge ja que ens permet fer èmfasi en metodologies i algoritmes en lloc de en operacions i eines matemàtiques elementals. Ara, moltes de les tasques mecàniques que l'alumne desenvolupava a mà, poden ser realitzades per mitjà d'aquests programes. És segur que la majoria de les calculadores i sistemes operatius del futur (ja molt proper) hauran d'incorporar el càlcul simbòlic.

1.2.3 Simulació

En molts camps de la Ciència, el gran progrés de la Informàtica durant les darreres dècades, no tan sols ha facilitat la realització de molts càlculs, sinó que ha donat lloc a una

[1] Per exemple: ningú no intentaria resoldre amb un llapis i un paper un sistema lineal de 15 equacions amb 15 incògnites. Un desenvolupament d'un determinant d'aquesta dimensió donaria una expressió amb 15! sumands, és a dir, més d'un bilió de sumands. Si cadascú d'aquests sumands ocupés 1 cm. i una llibreta tingués 100 pàgines de 30 línies de 15 cm. necessitaríem unes 30 milions de llibretes per aquest desenvolupament. A 1 segon per sumand es necessitaria uns 40.000 anys. No obstant això, qualsevol amb un simple ordinador personal i un senzill programa de càlcul pot efectuar aquesta operació en menys d'una dècima de segon.

tercera via d'aproximació al món físic, intermèdia entre la teoria i l'experiment, anomenada *Simulació*. Si es vol donar una imatge actual de la Ciència, és indispensable iniciar-se en aquesta nova metodologia.

Aquesta nova via consisteix en introduir a l'ordinador els elements essencials d'un model amb un mínim d'anàlisi. La potència de càlcul de l'ordinador permet comprovar si aquest model dóna la resposta esperada. Els resultats poden interpretar-se com els d'experiments ideals per estudiar sistemes modelitzats.

Algunes de les tècniques de simulació emprades avui dia poden ser fàcilment adaptades als temes més elementals de la Física, Química, Tecnologia, Economia, Sociologia, Biologia o Matemàtiques. En concret, citaré dos tipus de tècniques que es poden adaptar fàcilment a aquells nivells:

- *La resolució numèrica d'equacions del moviment*, que permeten resoldre molts tipus problemes físics (no només de dinàmica): *òrbites de planetes*, *circuits elèctrics, termodinàmics...* Aquesta tècnica es pot entendre i es pot aplicar sense coneixements d'equacions diferencials.

- Tècniques anomenades *models de joguina ("toy models")*: *autòmats cel·lulars, "random walks", algoritmes genètics, etc.* Aquestes tècniques apareixen sobretot en la descripció dels anomenats *Sistemes complexos*. Com el nom indica, són tècniques molt fàcils que, no obstant això, han permès entendre, en una petita part, sistemes tan complexos com la pròpia vida.

Les simulacions per ordinador tenen un gran nombre de característiques que les fan molt eficients per aplicar-les a l'ensenyament de les Ciències:

- Cada nou tipus de mesura de laboratori requereix un nou aparell de mesura. En simulació per ordinador, la mesura d'una nova variable requereix tan sol algunes línies més de codi.

- Una part important del disseny i preparació de la simulació es pot fer a l'aula normal, sense necessitat d'anar a l'aula d'informàtica (ni al laboratori).

- L'anàlisi de dades d'un experiment real necessita normalment un traspàs d'aquestes a un ordinador. En simulació, l'anàlisi de les dades és una de les parts del codi i, per tant, la simulació i l'anàlisi de dades es pot fer tot alhora.

- Amb tècniques de simulació podem experimentar amb propietats de sistemes que, com les propietats microscòpiques, són impossibles de controlar en un laboratori. També podem simular sistemes que no es poden portar al laboratori i canviar alguns paràmetres de l'experiment per entendre millor el significat d'aquests, encara que aquests valors dels paràmetres no es puguin donar en la realitat.

- En una simulació, estudiar les mateixes propietats de sistemes diferents representa només un canvi dels paràmetres del model.

- La possibilitat de fer una animació amb els resultats trobats de sistemes de moltes partícules ajuda a adquirir una intuïció de conceptes quotidians poc intuïtius, com la temperatura, la pressió, escalfar, refredar, la difusió, etc.

- Aspectes com l'espai i el temps es poden controlar en una simulació de tal forma que es pot *visualitzar* i analitzar en un temp raonable tant els fenòmens microscòpics que tenen lloc en fentosegons com els fenòmens astronòmics que triguen milers d'anys en completar-se.

1.2.4 Recopilació i anàlisi de dades

Els ordinadors són presents en totes les fases d'un experiment de laboratori, des del disseny de l'aparell a la recopilació i anàlisi de les dades obtingudes. Això permet, no només permetre a l'experimentador que pugui anar-se'n a dormir a la nit mentre l'ordinador va enregistrant els resultats de l'experiment, sinó que fa possible el disseny d'experiments que d'altra forma serien impossibles de realitzar-se.

Pràcticament tots els instruments de laboratori estan dissenyats per poder connectar-los a un ordinador. Els resultats de les mesures de l'experiment s'introdueixen de forma automàtica a la memòria de l'ordinador i en aquest, amb programes d'anàlisi numèric i de tractament estadístic, es processen i donen els resultats d'una forma més elaborada (per exemple, en forma de gràfics). L'anàlisi d'un conjunt de 1.000.000 de dades, per exemple, seria una tasca literalment impossible sense l'ajuda de l'ordinador. Comptar amb aquesta ajuda ens ha permès millorar notablement la precisió dels resultats dels experiments.

2. Objectius del treball

Els objectius del treball són els següents:

- Detectar parts del currículum de les matèries de Ciències Experimentals, Matemàtiques i altres disciplines on sigui possible introduir l'ordinador, no com una eina de consulta, sinó com una eina de càlcul i simulació.

- Introduir aspectes de càlcul numèric, algorísmica i tècniques de simulació adaptats als nivells d'Educació Secundària Obligatòria i Batxillerat per poder aplicar aquesta metodologia.

- Desenvolupar propostes per fer petites investigacions o treballs de recerca fent servir la simulació per ordinador.

- Realitzar activitats de foment de l'ús de l'ordinador per a resoldre problemes.

- Recopilar, seleccionar i ordenar recursos didàctics existents en l'àmbit d'aquest projecte.

3. Desenvolupament del treball

El desenvolupament del treball ha tingut diferents fases:

- Recol·lecció d'informació i formació personal.

- Creació de propostes de Treballs de Recerca i Petites Investigacions, algunes d'elles molt desenvolupades: *Els algorismes genètics, el problema del viatjant, Els nombres aleatoris, Simulació del tràfic mitjançant un autòmat cel·lular unidimensional i un altre bidimensional, Criptologia*.

- Propostes d'altres temes per Treballs de Recerca.

- Creació de l'entorn programable ***Simula***, un entorn de desenvolupament de simulacions de sistemes dinàmics formats per una partícula. Redacció de la descripció detallada del codi de l'entorn ***Simula*** i del seu funcionament.

- Creació i redacció de diferents aplicacions amb l'entorn ***Simula***: *Estudi del pèndul, Estudi de les Oscil·lacions lineals, Estudi dels Moviments de Planetes, Estudi de les forces de fricció, Estudi del tir parabòlic.*

- Organització de la ***1ª Pre-Olimpíada Informàtica Catalana***.

- Preparació de material didàctic d'entrenament i exercicis per les diferents fases d'aquesta competició.

- Preparació d'un curs de Fonaments de Programació i Llenguatge de programació C/C++ destinat al professorat d'Educació Secundària i en el qual es fomenta la utilització dels ordinadors per resoldre problemes dels diferents currículums de secundària.

3.1 Recopilació d'informació i formació personal

Per la preparació d'aquest treball va ser necessari consultar bibliografia i recursos existents referents a l'ús de l'ordinador per resoldre diferents tipus de problemes. Alguns llibres de mètodes numèrics donen idees fàcilment aplicables a nivells no universitaris.

Altres llibres clàssics de Física General com [Eisberg] i [Feynmann] proposen exercicis que s'han de resoldre mitjançant una calculadora programable o bé amb un ordinador.

Alguns llibres específics de simulació com [Gould] o [Trullàs] expliquen detalladament com analitzar diferents problemes amb l'ús de tècniques de simulació. En el primer d'aquests llibres es pot trobar el codi en llenguatge TrueBASIC de molts exercicis de simulació. Aquests codis són tots molt curts i fàcils d'entendre.

Entre la col·lecció d'informació recopilada sobre el tema és remarcable el *Curso Interactivo de Física en Internet* creat per Ángel Franco García de l'Escuela Universitaria Técnica Industrial d'Eibar. Aquest curs conté nombrosos *applets* de JAVA amb simulacions de diferents temes de Física i, el que és més important, un curs de JAVA i mètodes numèrics per tal de crear les nostres pròpies simulacions.

3.2 Propostes de Treballs de Recerca i Petites Investigacions

Aquesta col·lecció de 4 propostes totalment desenvolupades pretén donar exemples de Treballs de Recerca per a alumnes de Batxillerat. Cadascuna de les propostes podria ser un treball complet o dividir-se en diferents parts, segons el nivell de l'alumne. En aquestes propostes podem trobar una introducció teòrica al tema, una explicació de les tècniques de simulació necessàries i aspectes interessants per investigar. Aquestes propostes acaben amb un qüestionari que pot servir com a guia pel treball pràctic a desenvolupar.

Els títols de les propostes són:

3.2.1 Aplicació d'un algoritme genètic per resoldre el problema del viatjant

En aquesta proposta es presenta la tècnica de simulació dels *Algorismes genètics* proposada per Holland en 1986. Aquesta és una tècnica computacional de resoldre problemes simulant el procés natural d'evolució utilitzant la idea darwiniana de la selecció d'acord amb l'actitud.

Un algorisme genètic consisteix en generar una població inicial de possibles solucions a un problema. Cadascuna d'aquestes possibles solucions té assignat un valor segons una *funció d'aptitud* que és diferent en cada cas i representa la bondat de la solució.

El problema del viatjant és un dels problemes clàssics no resolt analíticament i que es pot estudiar mitjançant un algorisme genètic. El problema és el següent:

Un viatjant ha de visitar totes les ciutats d'un territori només una sola vegada i ha de tornar a la ciutat de partida. Es coneix les distàncies entre totes les parelles de ciutats. El problema consisteix en trobar l'itinerari amb distància mínima.

En aquesta proposta s'explica com implementar la tècnica dels Algorismes genètics en aquest problema en particular. Les qüestions de la part *Petites investigacions* permeten entendre tant el problema del viatger, com la forma d'implementar la tècnica en aquest problema.

3.2.2 Criptologia: xifrar i desxifrar missatges secrets

La *Criptologia* és la ciència de comunicar-se secretament. El principi bàsic de la Criptologia és modificar el missatge original entre l'emissor i el receptor de forma que sigui incomprensible per tota persona distinta d'aquests dos.

Podem dir que la Criptologia es pot dividir en dos parts: *Criptografia* i *Criptoanàlisi*.

La Criptografia es la part de la Criptologia que tracta del diseny i implementació de sistemes secrets.

El Criptoanàlisis és la part que es dedica a "trencar" aquests sistemes.

En aquesta proposta es fa una revisió dels mètodes clàssics de encriptació per substitució i el seu trencament mitjançant mètodes de freqüència.

3.2.3 Els nombres aleatoris

Les seqüències aleatòries són imprescindibles en molts tipus de simulacions. De fet, gairebé tots els llenguatges de programació incorporen alguna funció que generi nombres "aleatoris". En aquest projecte es tracta de reflexionar sobre què vol dir nombre aleatori, què tipus d'algorismes existeixen i fan servir els llenguatges de programació i quins tests podem passar a una seqüència de nombres aleatoris per assegurar-nos que és realment aleatòria.

3.2.4 Simulació del tràfic d'una autopista mitjançant un autòmat cel·lular i del tràfic d'una ciutat mitjançant un autòmat cel·lular bidimensional

Un autòmat cel·lular és un sistema format per cel·les. Cada cel·la pot tenir només un nombre finit d'estats possibles. En cada pas de temps (discret), o cada generació, pot canviar l'estat de les cel·les d'acord amb un conjunt de regles basades en els estats actuals de la cel·la i dels seus veïns més propers.

Els autòmats cel·lulars són molt fàcils d'implementar amb un ordinador i permeten simular sistemes molt complexos com pot ser els sistemes ferromagnètics, el trànsit d'una autopista o el trànsit d'una ciutat.

En aquesta proposta es fa una revisió de com implementar un autòmat cel·lular unidimensional i un altre bidimensional. Amb el primer es proposa simular el trànsit d'una autopista per intentar comprendre el fenomen de les retencions. Amb el segon es proposa simular el trànsit d'una ciutat ideal del tipus Eixample de Barcelona.

3.3 Propostes d'altres temes per Treballs de Recerca relacionats amb la simulació

A més de les propostes més desenvolupades mencionades a l'apartat anterior i que es troben íntegrament als annexos d'aquest treball, es proposen un conjunt d'altres propostes menys desenvolupades i que podrien ser també temes per Treballs de Recerca.

3.3.1 8-puzzle

El joc conegut com "8-puzzle" consisteix en ordenar 8 cel·les desordenades en una graella de 3X3 cel·les. Per tal d'aconseguir aquesta ordenació es pot moure en cada pas una cel·la només a la posició que estigui buida. El problema consisteix en trobar un camí que vagi de la configuració inicial a la configuració final.

5	**4**	**1**		**1**	**2**	**3**
6		**9**	$\rightarrow$	**4**	**5**	**6**
7	**2**	**3**		**7**	**8**	

3.3.2 El joc del Mastermind

En aquest joc, cada jugador ha d'endevinar una combinació de colors oculta per un altre jugador, i l'endevina fent suposicions sobre la combinació. L'altre jugador contesta amb dos fitxes de colors blanques i negres. La fitxa blanca indica que el jugador ha encertat el color, però no la posició. La fitxa negra indica que el jugador ha encertat el color i la posició. En cada jugada, el jugador té tota la informació de les fitxes blanques i negres de les jugades anteriors i amb aquesta informació ha d'intentar endevinar la combinació de colors.
Aquest és un joc que es pot resoldre amb un algorisme genètic. Els diferents "cromosomes" són combinacions de colors que tenen una funció d'aptitud depenent de la compatibilitat amb les puntuacions anteriors.

3.3.3 El dilema del presoner

El *Dilema del presoner* és un joc en el qual s'enfronten dos jugadors que representen dos delinqüents que han estat arrestats i són interrogats per separat. El jutge els hi presenta a cadascú d'ells la possibilitat de confessar. Si tots dos confessen passaran tres anys tancats a la presó. Si cap dels dos confessa no hi ha proves suficients i només se'ls hi condemna a un any de presó a cadascú. Si un dels dos confessa i l'altre no, al primer se li deixa en llibertat i al segon se li condemna a 5 anys de presó. Totes les possibilitats es poden resumir en la següent taula:

	l'altre jugador	confessa	no confessa
jugador			
confessa		3	0
no confessa		5	1

Si analitzem les possibles actituds d'un jugador sembla evident que és més beneficiós confessar i delatar al company. Si el company confessa convé que el jugador confessi també per evitar els 5 anys de presó. Si el company no confessa també convé que el jugador confessi per tal d'evitar la presó. Sempre la millor opció és la de confessar i, per tant, no cooperar amb el company. Aquest argument és totalment correcte, individualment és millor confessar, però col·lectivament no. Si tots dos confessen s'han guanyat 3 anys de presó cadascú (un total de 6 anys). Si només confessa un, s'han guanyat 5 anys de presó. I si no confessa cap, només guanyen 2 anys de presó. Aquesta situació provoca la següent paradoxa: L'interès individual aconsella un comportament egoista i l'interès col·lectiu aconsella una actitud solidària.

Aquest dilema constitueix un bon model per estudiar processos de cooperació biològics i sociològics.

El *dilema del presoner iteratiu* consisteix en repetir aquest joc un nombre alt de cops. D'aquesta forma es pot estudiar les diferents estratègies com: *confessar sempre, no confessar mai, confessar o no a l'atzar, no confessar fins que l'altre confessi un cop, fer el mateix que ha fet l'altre jugador a la jugada anterior.*

Si es simulen totes aquestes estratègies per ordinador i se'ls fa competir en una confrontació a parelles es pot estudiar el comportament col·lectiu de tots els jugadors.

3.3.4 Comportament emergent. Autòmat cel·lular bidimensional que simula el moviment de les formigues caminant

Com a il·lustració de com es pot aplicar la simulació per comprendre comportaments complexos comentaré el senzill experiment virtual que va proposar Christopher Langton en la Universitat de Michigan:

Va crear a l'ordinador una "formiga virtual", assignant-li una sèrie de regles senzilles per determinar la forma en què aquesta respondrà al seu entorn.

L'entorn consisteix en cel·les de tres colors diferents. El comportament de la formiga només dependrà del color de la cel·la on es trobi.

També està previst que la formiga canviï sota certes condicions el seu entorn.

Les regles són:

- Si la formiga penetra en una cel·la blanca, continua desplaçant-se en la mateixa direcció que portava.

- Si la formiga s'introdueix en una cel·la blava, aquesta adoptarà el color groc i la formiga girarà a l'esquerra.

- Si la formiga s'introdueix en una cel·la groga, aquesta adoptarà el color blau i la formiga girarà a la dreta.

Es va deixar simultàniament vàries formigues en la mateixa graella i el seu comportament va resultar molt semblant al de les formigues reals: en primer lloc, van caminar sense rumb definit durant els primers instants i, a continuació, es van trobar unes a les altres i van formar una fila. L'aspecte més interessant d'aquest experiment consisteix en que Langton no havia programat explícitament a les formigues per a que presentaren un comportament social. Aquest comportament és el que es diu comportament "emergent".

Aquest exemple, el qual pot donar peu a força comentaris, recull molt bé la filosofia de la Simulació. Només s'han programat elements molts senzills del model de "formiga" i hem comprovat que aquests elements són els responsables del comportament emergent del conjunt de formigues. En el cas més conegut de simular la trajectòria d'un planeta que gira al voltant del sol, no tindria massa sentit imposar que el planeta segueixi una trajectòria el·líptica sinó que, el que s'ha d'imposar és la regla més senzilla que consisteix en el tipus de força que actua entre el planeta i el Sol i que ve determinat per la llei de Newton. Que la trajectòria sigui el·líptica ha de ser un dels resultats de la simulació.

3.3.5 El mètode de Newton per trobar arrels d'equacions reals i complexes. Visualització de la conca d'atracció d'una equació no lineal senzilla

Les equacions que es poden resoldre analíticament són només les anomenades equacions lineals i poques més. La majoria de les equacions i sistemes que es fan servir en moltes parts de la Ciència actual són equacions i sistemes **no lineals**. Moltes d'aquestes equacions i sistemes s'han de resoldre necessàriament amb mètodes numèrics. El mètode de Newton és el mètode més àmpliament utilitzat per aquest propòsit. Si una equació té més d'una solució, cada aproximació inicial ens donarà una de les solucions. El conjunt de possibles aproximacions inicials que ens dóna una solució concreta s'anomena *conca d'atracció de la solució*. La estructura d'aquests conjunts és d'enorme complexitat i bellesa, i només es pot contemplar si fent servir l'ordinador per estudiar-la.

El treball consistiria en fer un programa per visualitzar la conca d'atracció de l'equació complexa $z^3-1=0$.

Parts del treball

- Estudiar i entendre el Mètode de Newton pel càlcul d'arrels d'equacions.
- Utilitzant una calculadora o l'ordinador trobar la solució d'algunes equacions no lineals com exemple.
- Generalitzar el mètode de Newton per resoldre sistemes d'equacions no lineals. Resoldre algun exemple.
- Estudiar què són els nombres complexos (només en forma binòmica).
- Aprendre a fer servir el mètode de Newton per calcular l'arrel quadrada d'un nombre complex.

- Construir un programa d'ordinador que dibuixi les conques d'atracció d'alguna equació senzilla (per exemple $z^3-1 = 0$)

3.3.6 Estudi de l'equació logística

¿Poden les *matemàtiques* capturar el complex moviment atmosfèric, la turbulència d'una explosió o l'evolució d'un ecosistema? O, pel contrari, en els sistemes en els quals hi ha un nombre molt elevat de variables que afecten al sistema de forma no lineal és completament impossible qualsevol predicció fins al punt que *el lleu aleteig de les ales d'una papallona en la selva amazònica pot alterar, als pocs dies, el curs d'un huracà.* A aquestes preguntes pretén donar resposta la ***teoria del CAOS***. Com moltes disciplines matemàtiques, la *teoria del CAOS* pot estudiar-se amb diferents nivell de profunditat. Les idees fonamentals d'aquesta teoria no són difícils de captar per un estudiant de Batxillerat. Es pot obtenir fàcilment de senzills models matemàtics comportaments complexos que ens recorden els que s'ha plantejat al començament d'aquest paràgraf.

Un dels models més senzills on es poden observar comportaments caòtics és en ***l'equació logística***. Aquest model es deu al biòleg Robert May que, en 1976 el va introduir per tal d'explicar la evolució d'una població d'insectes en un ecosistema tancat.

El treball consistiria en estudiar aquest sistema dinàmic, tant analíticament com numèricament. Per aquesta tasca es farà ús de l'ordinador i un senzill programa.

Els conceptes que s'han de tractar en el treball són: *Sistema dinàmic, l'equació logística, punts fixos, punts estables i inestables, òrbites de període 2,3,..., diagrames de recurrència, diagrames de Feigenbaum,etc.*

3.3.7 Equació de Laplace

Si es coneix la distribució de càrregues elèctriques a l'espai és possible calcular el camp elèctric total degut a aquestes càrregues mitjançant la llei de Coulomb.

Si la distribució de càrregues elèctriques és suficientment simètrica, és molt més fàcil el càlcul fent servir la llei de Gauss i després calcular el camp a partir del fluix.

Quan es tracta de una distribució que no té suficient simetria és millor fer servir *potencials elèctrics escalars*: Es determina el potencial elèctric de cada càrrega, es calcula la suma de aquests potencials i, finalment, s'obté el camp diferenciant el potencial respecte la posició.

Si no es coneix la localització de les càrregues és possible en molts casos trobar el potencial i, per tant el camp, en una regió si es coneix el valor del potencial al llarg de les fronteres de la regió. Això es pot fer a partir de l'equació de Laplace.

L'equació de Laplace no és una llei independent, s'obté de la llei de Gauss i és, junt amb aquesta llei de Gauss, una reformulació de la llei de Coulomb.

L'equació de Laplace és: $\frac{\partial^2 V}{\partial x^2}+\frac{\partial^2 V}{\partial y^2}+\frac{\partial^2 V}{\partial z^2}=0$ o bé $\Delta V=0$.

L'equació de Laplace es pot interpretar dient que la funció V(x,y,z) en una regió lliure de càrregues ha de ser de tal forma que la suma de les seves corbatures al llarg de direccions mútuament perpendiculars (x,y,z) sigui 0.

Si simplifiquem la interpretació a dos dimensions, per a què una funció compleixi que

$\frac{\partial^2 V}{\partial x^2}+\frac{\partial^2 V}{\partial y^2}=0$ ha de complir que, o bé cadascuna de les derivades parcials sigui 0 (correspon a un pla), o bé que las derivades parcials tinguin signe diferent. Això vol dir que aquest punt té curvatura diferent segons la direcció (punt de sella)

En cap punt de la regió (lliure de càrregues) pot haver-hi un màxim o un mínim.

Una membrana elàstica entre sopors fixos té propietats anàlogues al potencial elèctric en una zona lliure de càrregues. De fet, la funció altura h(x,y) de la membrana satisfà l'equació de Laplace. Aquest símil funciona com un "computador analògic" que produeix la solució de l'equació de Laplace per als valors de V especificats a la frontera.

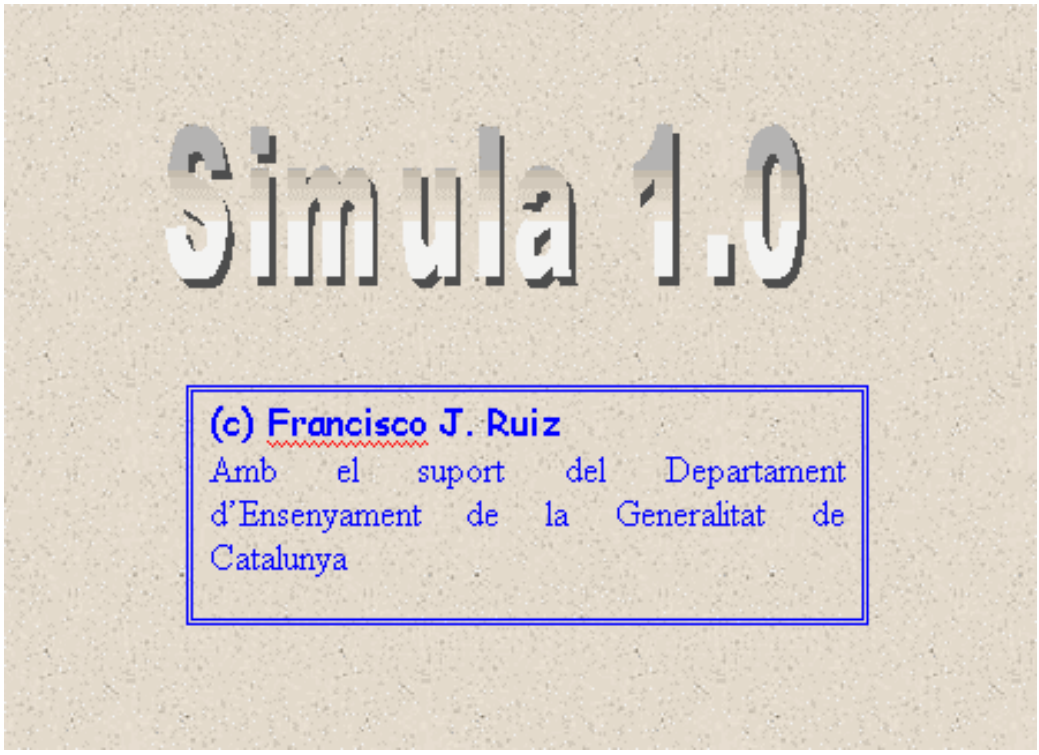
Per a molts sistemes de gran interès pràctic no existeixen solucions analítiques per a l'equació de Laplace. Exposarem un mètode numèric per a aquest fi.

És una conseqüència de l'equació de Laplace que: *El valor V(x,y) en qualsevol punt en una regió lliure de càrrega ha de ser el promig dels seus valors en quatre punts veïns localitzats simètricament.*

Per iniciar el càlcul s'assignen valors de V a tots els punts interiors de la xarxa (la resolució d'equacions diferencials implica sempre una solució de prova). Els valors finals no depenen dels inicials, encara que quant més raonables siguin aquests valors més ràpid serà el procediment.

	2	2	2	2				2	2	2	2	
1	1	1	1	1	1		1	1.25	1.25	1.25	1.25	1
1	1	1	1	1	1		1	1	1	1	1	1
1	1	1	1	1	1		1	1	1	1	1	1
1	1	1	1	1	1		1	0.75	0.75	0.75	0.75	1
	0	0	0	0				0	0	0	0	
	2	2	2	2				2	2	2	2	
1	1.31	1.38	1.38	1.31	1		1	1.36	1.44	1.44	1.36	1
1	1.06	1.06	1.06	1.06	1		1	1.08	1.11	1.11	1.08	1
1	0.94	0.94	0.94	0.94	1		1	0.92	0.89	0.89	0.92	1
1	0.69	0.63	0.63	0.69	1		1	0.64	0.57	0.57	0.64	1
	0	0	0	0				0	0	0	0	

3.4 L'entorn programable *Simula*.



Segons George Polya (*How to Solve it*), les fases per resoldre un problema són:

- Comprensió del problema
- Confecció d'una estratègia.
- Portar a terme l'estratègia.
- Revisió dels resultats.

Si fem servir l'ordinador per la tercera fase, l'estratègia ha de ser dissenyada a mida de l'ordinador. Ensenyar a l'ordinador com seguir un pla per resoldre un problema obliga a descriure'l detalladament amb un llenguatge concret, concís i formal. És sabut que en el moment en el qual intentem explicar a algú algun problema, ens assabentem si l'entenem o no. Ensenyar a l'ordinador permet entendre millor el problema. Per tant, el benefici que dóna l'ús de l'ordinador és doble: per una part ens ajuda a entendre millor el problema i, per l'altra, l'ordinador ens ajuda a solucionar aquest problema. Per poder gaudir d'aquest doble benefici no és suficient fer servir programes ja fets a mida per resoldre problemes concrets, sinó que és necessari fer servir programes molt oberts, que permetin definir completament el problema o, millor encara, que sigui l'alumne qui confeccioni completament el programa.

És intenció d'aquest treball i del seu autor fomentar que sigui l'alumne el que disseny el programa per permeti simular un sistema concret. Per cada possible sistema hi ha

possibilitats de trobar programes i *applets* que funcionen com un videojoc. L'usuari pot manipular només uns quants paràmetres però la relació que hi ha entre aquests paràmetres i el model que es fa servir a la simulació del sistema ens sembla un misteri.

És obvi que, amb l'estat actual de l'ensenyament de la Programació a l'ensenyament pre-universitari, és convenient fer un pas intermedi entre que l'alumne escrigui tot el codi i que faci servir un programa ja fet. És per això que l'entorn ***Simula*** pot ser útil. Aquest entorn no és exactament un programa ja que està previst que l'alumne el modifiqui i el torni a compilar. De fet, el que s'ha fet és separar en arxius separats les parts que s'han de modificar per cada sistema a estudiar o les parts que determinen les variables a mesurar.

L'entorn ***Simula*** consisteix en els següents arxius.

simula.c codi escrit en C amb les funcions generals per tots els sistemes.

simula.h arxiu on estan les definicions de les variables i funcions del programa.

forca.h arxiu que conté la descripció en codi C del sistema a simular.

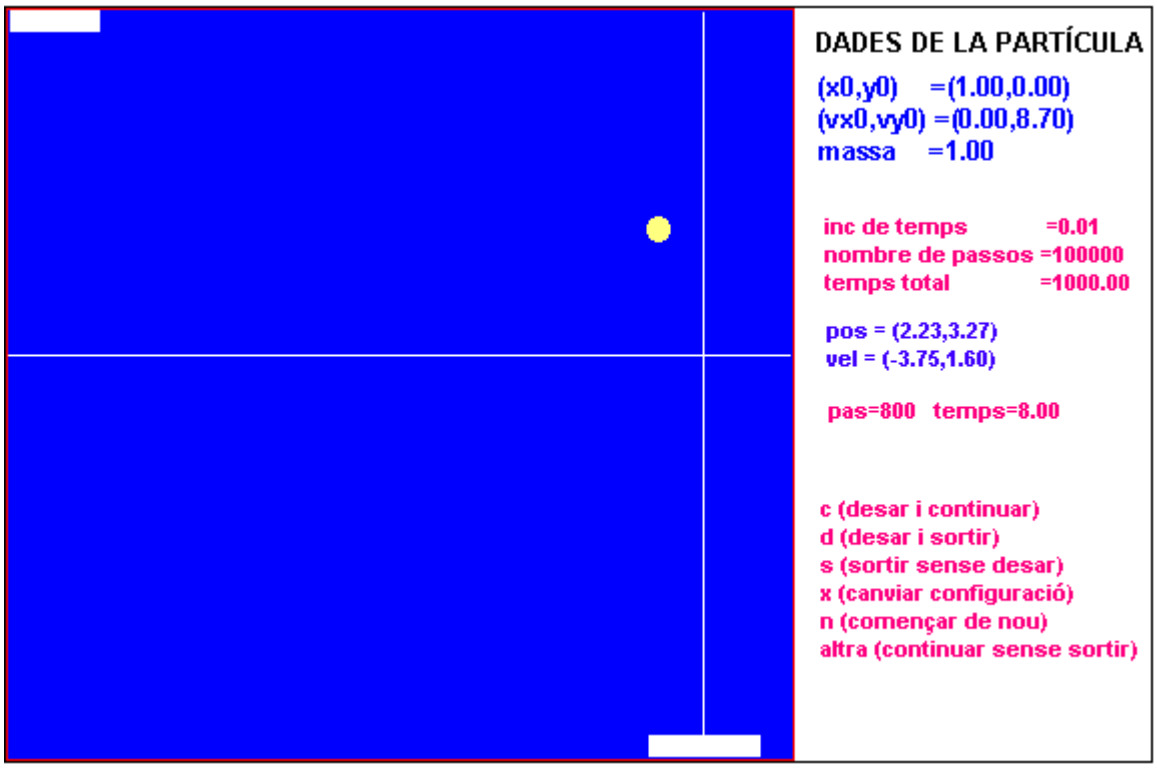
comprova.h arxiu que conté la descripció en codi C de les condicions que s'han de donar per tal de que s'aturi temporalment la simulació. Això és important per poder desar en disc les dades en aquest instant concret.

in.txt arxiu que conté les dades de les condicions inicials del sistema i de la simulació.

out.txt arxiu que conté les dades seleccionades de sortida. Aquest arxiu pot tenir un nom diferent. El seu nom es determina com un paràmetre en l'arxiu *in.txt*. Aquest arxiu és útil per tal de fer una anàlisi posterior de les dades amb altre programa, com pot ser un full de càlcul.

3.4.1 Breu descripció del funcionament de *Simula*:

La pantalla principal del programa és tal i com mostra la següent figura:



Hi ha una zona rectangular que és l'àrea de dibuix on es veurà el moviment de la partícula. En aquesta àrea es mostra les coordenades lògiques dels vèrtexs superior esquerre i inferior dreta. A més, es mostra també els dos eixos de coordenades (x=0 i y=0). Tant el color com la mida de l'àrea de dibuix i del cercle que representa la partícula es pot modificar.

En aquesta pantalla es mostra les dades inicials de la partícula: posició i velocitat inicial, així com la seva massa. Aquestes dades es mostren en color blau.

A continuació, i de color vermell, es mostra les dades temporals fixes:

- Increment de temps: (valor de Δt) necessari en la discretització de les equacions del moviment.
- Nombre total de passos: màxim nombre de passos abans que s'aturi definitivament la simulació. Aquest valor es pot posar molt gran ja que la simulació es pot aturar d'altres formes.
- Temps total màxim de la simulació. Aquest temps total no correspon al temps real de duració de l'execució de la simulació, sinó el temps que pretén simular.

Les dades que es mostren a continuació són les dades que canvien cada pas de temps. En blau, la posició i velocitat instantània de la partícula i en vermell, el pas de temps actual i el temps de simulació actual. S'ha de tornar a insistir que aquest temps no correspon al temps d'execució del programa.

Per aturar la simulació es pot fer amb la tecla ESC. Després d'aturar la simulació apareix un menú que permet:

- Desar la configuració instantània actual i continuar (opció c)
- Desar la configuració i sortir (opció d)
- Sortir sense desar la configuració (opció s)
- Canviar la configuració (opció x)
- Començar de nou la simulació (opció n)
- Continuar sense desar (altra tecla)

Els paràmetres que es poden canviar amb l'opció ***canviar la configuració*** són:

- Les coordenades lògiques de l'àrea de dibuix (lox, loy, lfx, lfy).
- El pas de temps (t).
- El nombre total de passos (nombre_passos)
- El control de pauses (pause i passos_pause). Si el control pause=1 cada passos_pause passos s'aturarà la simulació.
- El control de retard (retard) controla la velocitat de visualització de la simulació.
- El control de la trajectòria (traj) controla si es traça la trajectòria o només la posició de la partícula.
- L'algorisme d'integració utilitzat en la simulació entre Euler, Euler-Richarson i Verlet.
- La posició i velocitat inicial de la partícula (px0,py0,pvx0,pvy0).
- La massa de la partícula (pmassa).
- La mida del cercle que representarà la partícula en píxels (pradi)

A més d'aquests paràmetres, el fitxer *in.txt* conté altres paràmetres de configuració de la simulació.

Als annexos II i III es descriu amb més detall aquest entorn.

3.5 Creació i redacció de diferents aplicacions amb l'entorn *Simula*

Per tal d'il·lustrar l'ús de l'entorn ***Simula*** s'ha redactat un conjunt de 5 aplicacions detallades completament resoltes.

Els títols d'aquestes aplicacions són:

- *Estudi del pèndul simple.*
- *El tir parabòlic.*
- *Estudi de les Oscil·lacions lineals.*
- *Estudi dels Moviments de Planetes.*
- *Estudi de les forces de fricció..*

3.5.1 Estudi del pèndul simple

Un pèndul simple és un sistema format per un cos sòlid connectat amb una vareta o una corda. L'altre extrem de la vareta és fix. El cos està limitat a moure en una circumferència de radi igual a la longitud de la vareta i centre el punt fix de la vareta. El sistema s'idealitza amb les següents consideracions:

La massa de la vareta és menyspreable.
Les dimensions del cos són menyspreables en comparació amb la longitud de la vareta.
No hi ha fregament.

Amb l'ajuda de *Simula* s'estudia:

El període del pèndul en funció de la seva longitud
El període del pèndul en funció del valor de g
El període del pèndul en funció de la seva amplitud

3.5.2 El tir parabòlic

Sota la influència de la gravetat, els cossos canvien la seva velocitat verticalment però no horitzontalment. Això produeix una composició de moviments: un moviment uniformement accelerat en la direcció vertical i un moviment uniforme en la direcció horitzontal. La trajectòria produïda per aquesta composició de moviments és una paràbola.

L'estudi d'aquest tipus de moviment es pot fer analíticament i, per tant, és un bon sistema per a comprovar els resultats analítics i numèrics.

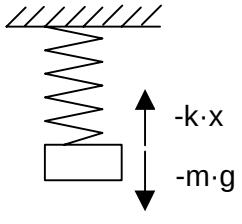
En aquesta aplicació s'estudia la relació entre l'abast i alçada màxima en funció de la velocitat inicial, així com la relació entre l'abast i l'alçada màxima en funció de l'angle

3.5.3 Estudi de les Oscil·lacions lineals

Per tal que existeixi una oscil·lació, és necessari que hi hagi una força oposada al augment de la coordenada corresponent. El cas més senzill i el que estudiarem és el cas de les *oscil·lacions lineals*, en les quals, aquesta relació entre la força i l'augment de la coordenada corresponent és proporcional, és a dir:

$$\vec{F} = -k \cdot \vec{x}$$

Aquesta és una forma de la llei de Hooke i representa bé la força recuperadora d'una molla que es desplaça lleugerament de la seva posició d'equilibri. Si el desplaçament inicial en una molla és més gran, la fórmula de Hooke ja no és certa i les oscil·lacions (en el cas que hi hagi) ja no seran lineals.



En aquesta aplicació s'estudia aspectes com l'amplitud, el període o l'esmorteïment i la seva relació amb la gravetat o les condicions inicials

3.5.4 Estudi dels Moviments de Planetes

Les lleis de Kepler van ser fonamentals per tal que Isaac Newton (1642-1727) formulés que el moviment de tots els planetes és degut a una força central, la força de gravitació, que té la següent forma:

$$F = G \cdot \frac{m_1 \cdot m_2}{r^3} r$$

En aquesta interessant aplicació es descriu la forma de comprovar la tres lleis de Kepler mitjançant la llei de gravitació de Newton.

3.5.5 Estudi de les forces de fricció.

Quan un cos es mou a una velocitat relativament petita a través d'un fluid tal com un líquid com pot ser l'aigua o un gas com pot ser l'aire, la força de fricció es pot obtenir aproximadament suposant que és proporcional a la velocitat i oposada a ella, és a dir:

$$\vec{F}_f = -B \cdot \vec{v}$$

Aquesta constant B depèn de la geometria del cos i de la viscositat del fluid.

A mesura que augmenta la mida i/o la velocitat del cos, el fluix del fluid es torna turbulent i aquesta turbulència dóna lloc a una força de fricció més gran. Aquesta força s'aproxima molt bé si es pren proporcional al quadrat de la velocitat:

$$F = b \cdot v^2$$

Amb el primer model podem simular el comportament d'una gota d'aigua que cau del cel. Amb el segon model podem simular el comportament d'un cos més pesat, com ara un paracaigudista.

3.6 Primera Pre-Olimpíada Informàtica Catalana

Durant el curs 2000-2001 vaig col·laborar juntament amb la Subdirecció General de Tecnologies de la Informació en la Organització i de la *Primera Pre-Olimpíada Informàtica Catalana* dirigida als alumnes d'ESO i primer curs de Batxillerat i Cicles Formatius. En aquesta organització s'ha tingut la col·laboració de la Comissió Organitzadora de l'*Olimpíada Informàtica Espanyola*, comissió coordinada per la Fundació Aula i integrada per les Facultats d'Informàtica de les Universitats

Politècniques de Madrid i Barcelona, les Facultats d'Informàtica de Deusto, la Corunya, Granada i Màlaga.

La *Pre-Olimpíada Informàtica Catalana* té per objectiu estimular l'aprenentatge de l'algorísmica i la programació entre els joves estudiants catalans, atès que aquestes disciplines poden constituir un ajut i un estímul important en la formació intel·lectual de l'estudiant, i constituiran, sens dubte, coneixements útils en el futur. Alhora, pretén ésser un primer pas per a preparar l'eventual participació en activitats i competicions de més complexitat, com l'Olimpíada Informática Espanyola – organitzada per la Fundació Aula - o "La Olimpiada Informática Internacional".

La meva col·laboració en aquesta competició ha consistit en les següents fases:

- Redacció, juntament amb la resta del comitè organitzador, de les bases de la *Pre-Olimpíada*.

- Redacció d'una col·lecció d'enunciats d'exercicis d'entrenament amb algunes solucions.

- Cerca d'informació d'altres *Olimpíades Informàtiques* i competicions similars nacionals i internacionals: forma de funcionament, el nivell de dificultat,...

- Cerca d'informació i difusió d'aquesta informació respecte a l'obtenció i instal·lació dels diferents compiladors de llenguatges admesos en la competició.

- Disseny i manteniment d'una pàgina WEB de la competició. En aquesta pàgina es pot trobar:
  - la presentació;
  - les bases;
  - informació referent als diferents compiladors;
  - informació de referència d'altres competicions similars nacionals i internacionals;
  - exercicis d'entrenament amb indicació de nivell de dificultat i algunes solucions en diferents llenguatges de programació;
  - enunciats, arxius de prova, solucions i resultats de la fase prèvia de la competició;
  - enunciats, arxius de prova, solucions i resultats de la fase final de la competició;

- Col·laboració en la fase d'inscripció.

- Redacció, juntament amb la resta del comitè organitzador, d'un conjunt d'exercicis i selecció dels definitius per a la fase prèvia. Cada exercici disposa d'un conjunt de 10 jocs de prova per a la correcció automàtica. Cada un dels jocs de prova intenta comprovar la superació de diferents dificultats de l'exercici.

- Construcció dels programes informàtics necessaris per a la correcció automàtica dels exercicis resolts pels concursants d'aquesta fase prèvia.

- Correcció i, juntament amb la resta del jurat, confecció de la classificació de la fase prèvia.

- Difusió de la classificació de la fase prèvia així com dels jocs de prova i les solucions dels exercicis.

- Redacció, juntament amb la resta del comitè organitzador, d'un conjunt d'exercicis i selecció dels definitius per a la fase final.

- Preparació dels ordinadors de la seu de la fase final.

- Assistència durant la realització de la fase final.

- Organització prèvia general.

- Resolució de dubtes.

- Recopilació dels arxius dels concursants.

- Correcció i, juntament amb la resta del jurat, confecció de la classificació de la fase final.

- Difusió de la classificació final.

3.7 Curs de Fonaments de programació. Entorn C/C++

Com a complement dels materials realitzats per fomentar la realització de Treballs de Recerca amb l'ordinador, he dissenyat amb la col·laboració de Lluís Lores un curs de Fonaments de programació per a professors de Secundària.

Aquest és un curs eminentment pràctic en el qual s'introduirà simultàniament tècniques de programació elemental i la sintaxi del llenguatge C/C++ en l'entorn Visual C++. El curs se centra en la programació estructurada i modular i no entrarà pas en la *programació orientada a objectes* ni en la programació en l'entorn *Windows*. S'ha tingut un interès especial de no fer un simple manual de C/C++, dels quals hi ha molts i més complets que aquest. Aquest curs se centra en la realització de pràctiques on es va introduint conceptes generals de programació i específic de C/C++. Gairebé tots els programes proposats, tant els de les pràctiques com els dels exercicis, tenen una característica comú: són molt curts. Penso que aquesta característica és bona qualitat per a un curs d'aprenentatge. Encara que els codis de la majoria de les aplicacions que es poden fer estan formats per moltes (centenars o milers) línies, la majoria d'aquestes fan referència a aspectes de presentació, comprovació de dades i d'errors i documentació, molt necessàries per a una aplicació final, però que fan més difícil la comprensió de la part fonamental del programa: l'algorisme. Una vegada que l'alumne aprengui aquesta part fonamental, és necessari que complementi els seus programes amb els elements necessaris per donar-li el nivell d'acabament adequat.

3.7.1 Estructura del curs

El curs és un curs telemàtic que s'emmarca dins l'oferta que anualment proposa el Departament d'Ensenyament. S'estructura en 8 mòduls i cada mòdul està pensat per

dedicar-hi dues setmanes. Consta d'un resum teòric, un conjunt de 6 a 9 pràctiques i uns exercicis. A més, la majoria de mòduls contenen també alguna pràctica addicional d'ampliació i exercicis relacionats també d'ampliació que no són de lliurament obligatori.

3.7.2 Resum teòric

En cada mòdul es troba una pàgina amb un resum de tots els elements que es tractaran en el mòdul. Està redactat com un manual amb exemples molt curts i està pensat sobretot per consultar elements concrets del llenguatge C/C++.

3.7.3 Pràctiques

En tots els mòduls es poden trobar de 6 a 9 pràctiques, més algunes d'ampliació. A cadascuna d'aquestes pràctiques hi podem trobar un o dos programes comentats en els quals es fa servir un nombre reduït d'elements del llenguatge treballats en el mòdul.

Podem dir que hi ha dos tipus de pràctiques repartides en tots els mòduls:

- Pràctiques d'exemples directes d'ús d'algun element del llenguatge, per entendre bé el seu funcionament.
- Pràctiques d'aplicacions a altres camps: geometria, probabilitat, estadística, física, economia, informàtica, teoria de jocs, criptografia, mètodes numèrics, etc.

En molts mòduls hi ha alguna pràctica d'ampliació que, o bé utilitza aspectes avançats del llenguatge, o bé s'utilitzen elements coneguts en situacions no immediates. En qualsevol cas, aquestes pràctiques d'ampliació es poden ometre sense perdre el fil del curs. Tots els codis de les pràctiques es poden trobar com a material del curs i, per tant, no és necessari tornar a escriure'ls. Sí que es demana a l'alumne que faci la compilació i la construcció del fitxer executable, i que provi de fer l'execució del programa en diverses situacions, així com fer lleugeres modificacions al codi font i tornar a construir l'executable per veure les conseqüències d'aquestes modificacions.

3.7.4 Exercicis

En tots els mòduls hi ha una proposta d'entre quatre i sis exercicis. Aquests exercicis són els elements que permeten al tutor del curs fer un seguiment avaluatiu de l'alumnat. Normalment consisteixen en aplicacions que suposen lleugeres modificacions de les pràctiques del mòdul o noves aplicacions basades en aquestes. També, en alguns mòduls es pot trobar algun exercici d'ampliació. Aquests exercicis no són obligatoris però el tutor els guiarà i corregirà també en el cas que l'alumnat els lliuri.

4. Conclusions

Aquest treball ha suposat una recopilació i construcció de projectes en els quals l'ordinador no sigui només una eina de consulta sinó una vertadera eina de càlcul i simulació. Pot ser, siguem molts els professors interessats en la programació que volem alguna excusa per introduir aquests coneixements en els nivells de Secundària Obligatòria i Batxillerat. Aquestes propostes de Treballs de Recerca i Petites Investigacions que acompanyen al treball poden ser aquesta excusa i, pot ser, la necessitat d'adquirir aquests coneixements per poder desenvolupar aquestes propostes faci que algú torni a demanar la incorporació de disciplines de programació informàtica en els currículums d'una forma seria i estructurada.

A més de l'elaboració d'aquest material he volgut construir un entorn programable que he anomenat ***Simula***. Aquest entorn és un pas intermedi entre donar un programa fet completament –en el qual l'alumne seria només un usuari que podria canviar el valor d'alguns paràmetres- i que l'alumni faci totalment el programa que pugui simular el sistema a estudiar. La manipulació de les "entranyes" d'aquest entorn serà necessari per podet treure profit a aquest.

Com a complement i en la línia d'aquest treball vaig col·laborar de forma molt intensa en l'organització de la Pre-Olimpíada Informàtica Catalana. El resultat d'aquesta competició posa en evidència la manca de motivació de gran part de l'alumnat per la programació i per l'algorísmica. La consolidació d'aquesta Pre-Olimpíada serà un estímul per la creació de material que pugui superar en un futur proper aquesta situació.

La tercera activitat que he realitzat com a complement a la resta d'activitats relacionades amb el treball ha estat el disseny d'un curs de Fonaments de Programació per a professors de Secundària. Encara que el material d'aquest curs no pertany al meaterial elaborat per aquest treball, és un altre punt per tal d'impulsar que la Programació, l'Algorísmica i la Simulació es puguin consolidar com a part integrant del currículum de les matèries científiques del Batxillerat.

5. Bibliografia

General

Polya, George, *Cómo plantear y resolver problemas.* Ed. Trillas, 10ª ed. 1990.

Richard P. Feynmann, i altres. *The Feynmann Lectures on Physics,* Addison-Wesley . 1963.

Eisberg, R.M., Lerner, L.S., *Física, Fundamentos y Aplicaciones.* Ed. McGrawHill 1986.

Ordinadors i educació

Catellsaguer, Q. (1986) "La informàtica a l'ensenyament de les matemàtiques", *Butlletí del Col·legi Oficial de Doctors i Llicenciats*, núm. 60, pp. 22-23.

King, D. (1986) "Programa abiertos". *Cuadernos de Pedagogía*, núm. 135 pp. 52-56.

Quintana, J. (1996) "Quines aplicacions informàtiques a l'ensenyament obligatori?". *Escola Catalana*, núm. 330, pp. 6-7.

Quintana, J. (1997) "Programas Informáticos en la Educación Secundaria". Aula de Inovación Educativa, núm. 67, pp. 24-25.

Gros, B. (1987): *Aprender mediante el ordenador. Posibilidades pedagógicas de la informática en la escuela*. Barcelona. PPU.

Mètodes numèrics en general

Guardiola R., Higón E. i Ros, Josep, *Mètodes numèrics per a la Física*. Universitat de València. 1995

Cohen, A.M. *Análisis numérico,* Ed. Reverté, 1977.

Grau Sánchez, M., Noguera Batlle, M. *Càlcul numèric*. Edicions UPC. 1993.

William H. Press i altres, *Numericals Recipes,* Cambridge University Press (1992).

Mètodes de simulació

Trullàs Simó, Joaquim, *Física bàsica amb ordinador.* Edicions UPC. 1993.

Gould H. y Tobochnik Jan, *An Introduction to Computer Simulation Methods. Applications to Physical Systems.* Ed Addison-Wesley. 2ª ed. 1996.

Paul L. DeVries, *A First Course in Computational Physics,* John Wiley & Sons. 1994

Ian R. Gatland, "Numerical integration of Newton's equations including velocity-dependent forces" *Am J. Phys.* ***62****, 259 (1994).*

Algorismes genètics

Holland, John H. "Genetic Algorithms", *Scientific American* 267, Julio de 1992, pp. 66-72.

P. Larrañaga . "Algoritmos Genéticos" Departamento de Ciencias de la Computación e Inteligencia Artificial Universidad del País Vasco – Euskal Herriko Unibertsitatea
http://www.geocities.com/CapeCanaveral/9802/3d5ca000.htm

Sutton, P i Boyden S, "Genetic algorithms: A general seach procedure", *Amer. J. Phys.* **62**, 549 (1994).

El problema del viatjant

Programa DEMO de "El problema del viajant" resol amb un algorisme genètic:
http://personal1.iddeo.es/complexsystems/Geneticos.htm

El dilema del presoner

Poundstone, William. *El dilema del prisionero*. Alianza Editorial. 1992.

Bass, Thomas A. y Martín, Luna. "Por qué ganan los buenos". Revista *Muy Interesante*, nº 203. Abril 1998.

Axelrod, Robert. *La evolución de la cooperación*. Alianza Universidad. 1996.

Hofstadter, Douglas R. Temas Metamágicos. Revista *Investigación y ciencia*. Agosto 1983.

Dawkins, Richard. *El gen egoísta*. Salvat Ciencia. 1994.

Kropotkin, Peter Alexeivich. *El apoyo mutuo*. Ediciones Madre Terra. 1970.

Teoria del Caos. Estudi de l'Equació Logística. Fractals. Conques d'atracció. Mètode de Newton. Autòmats cel·lulars

Martín, Miguel Ángel, Morán, Manuel, Reyes, Miguel (1995): *Iniciación al Caos*, Editorial Síntesis. Madrid

Castillo Enrique *et al.*(1993): *Mathematica,* Ed. Paraninfo. Madrid

Barrallo, Javier: (1993): *Geometría fractal. algorítmica y representación.* Anaya Multimedia. Madrid

Solé, Ricard V., Manrubia, S.C., *Orden y caos en sistemas complejos*. Edicions UPC. 1996.

Langton, Chris, editor, *Artificial Life,* Addison-Wesley (1989)

Langton, Chris, "Studying artificial life with cellular automata", Physica D **22**, 120 (1986).
Stauffer, D. "Programming cellular automata", *Computers in Physics* **5**(1), 62 (1991)

Criptologia

Singh, Simon: *Los códigos secretos*. Debate Ed.

Allan Poe,Edgar. *El escarabajo de oro* . Ed. Vicens Vives

Conan Doyle, Arthur. *Sherlock Holmes* (algunes de les seves aventures)

Simulació del tràfic

Sánchez, Á. :*Revista Española de Física,* **10** (4), 1996

Nagel, K: *Phys. Rev. E* **53**, 4655-4672 (1996)

Nagel, K i Schreckenberg: *J. Phys.* **1** (France) **2,** 2221-2229 (1992)

Stauffer, D.: *Computers in Physics* **10**, 341-348 (1996)

http://www.theo2.physik.uni-stuttgart.de/helbing/RoadApplet/

Cuesta, J. A., F.C. Martínez, J.M. Molera i A. Sánchez: *Phys. Rev.* E **48**, R4175-R4178 (1993)

SÁNCHEZ, A.: *Revista Española de Física* **10** (4) (1996)

Joc del Mastermind genètic

http://geneura.ugr.es/~jmerelo/GenMM/GenMM.es.shtml

J. J. Merelo, J. Carpio, P. Castillo, V. M. Rivas, G. Romero GeNeura Team, "Finding a needle in a haystack using hints and evolutionary computation: the case of Genetic Mastermind"
http://geneura.ugr.es/~jmerelo/newGenMM/index.html

Programació en C/C++

Ruiz, F.J., Lores, Ll. *Fonaments de programació. Entorn Visual C++.* SGTI

Pappas, C.H., Murray, W.H. *Visual C++ 6.0. Manual de referencia*. Osborne McGraw-Hill. 1999

Gascón Sánchez M.F., García-Bermejo Giner, J. R.*Visual C++ 6. Guia en 10 minutos o menos*. Prentice Hall. 1999

Zaratian Beck. *Microsoft Visual C++ 6.0 Manual del programador*. Microsoft Press. McGrawHill. 1999

Schildt, H. *Programación en Turbo C*. Borland-Osborne/McGraw-Hill 1988

Schildt, H. *Turbo C. Programación avanzada*. Borland-Osborne/McGraw-Hill 1990

B. Kernighan y D. Ritchie.*El lenguaje de programación C* (2ª ed.). Prentice-Hall. 1991

Stroustrup, Bjarne.*El lenguaje de programación C++* **3ª ed.**Addison-Wesley Turpial, 1997

http://cfp401.freeservers.com/cursos/c1/c.htm
http://proton.ucting.udg.mx/tutorial/c/
http://www.crysoft.com/cursos/endetalle.asp?pide=C (molt elemental)
http://www.uib.es/c-calculo/cursc.htm
http://www.elrincondelc.com/ (es pot trobar codi font de molts exemples)

Recursos generals d'INTERNET

Pàgines amb recursos per a l'ensenyament i aprenentatge de les ciències de la naturalesa: *http://www.xtec.es/recursos/ciencies/index.htm*

Aquí matemàtiques!. Informacions i propostes matemàtiques per a alumnes d'ESO i Batxillerat: http://www.xtec.es/recursos/mates/aqui/index.htm

El país de la tortuga, Introducció al món del Logo.:http://www.xtec.es/logo/index.htm

La baldufa: *http://baldufa.upc.es*

Software Teaching of Modular Physics, Surrey University, UK:
http://www.ph.surrey.sc.uk/stomp.

Learning Aproach to Physics, Open University, UK:
http://yan.open.ac.uk/flap/FLAPHom.html,

Instructorial Technology Program, Baekeley University

Franco García, Angel, *Física con ordenador: Curso Interactivo de Física en Internet.* Escuela Universitaria de Ingeniería Técnica Industrial de Eibar:
http://www.dfa.ua.es/~agm/index.html/

6. Agraïments

Voldria mencionar a algunes persones que m'han ajudat a preparar-me per realitzar aquest treball i que m'han ajudat directament en la seva realització.

Al professor Quim Trullàs, del Departament de Física i Enginyeria Nuclear de la Universitat Politècnica de Catalunya, per introduir-me en tècniques de simulació, algunes de les quals contemplo en aquest treball; per aportar directa i indirectament idees brillants i suggeriments al projecte; i per oferir-se amablement a la co-tutorització del mateix.

A Núria Agell, del Departament de Mètodes Quantitatius a ESADE de la Universitat Ramon Llull, per confiar en mi i voler empentar-me cap a la investigació en Intel·ligència Artificial.

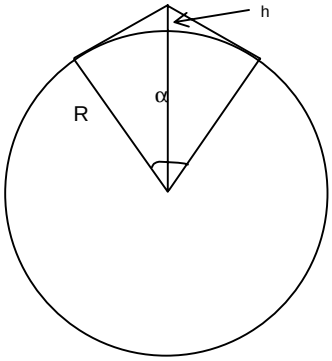
A Santiago Manrique, de la Subdirecció General de Tecnologies de la Informació per dirigir i tutorar aquest treball i per proposar-me activitats paral·leles directament relacionades amb ell.

Annex I. Exemples de problemes d'aplicació

Exemple 1: Veiem un exemple en el que podem il·lustrar la necessitat de programes de càlcul per resoldre problemes senzills.

Suposem que la Terra fos una esfera perfecta de radi R= 6.367 x 10 ^{6} m. Imaginem que tenim una corda que envolta perfectament la superfície d'aquesta per un cercle màxim. La longitud d'aquesta corda hauria de ser L = 2πR = 40.005.040,85 m. Si afegim 1 metre més a aquesta corda, això permetria que per un punt d'aquesta circumferència màxima es pogués aixecar la corda tal i com indica el dibuix. Anem a calcular l'alçada ***h*** *que permet aquest afegit.*

Amb eines trigonomètriques senzilles podem trobar que les magnituds α, h i d (veure dibuix adjunt) estan relacionats per les equacions:



$$(R+h)^2 = d^2 + R^2$$

$$d = R \cdot \mathrm{tg}\,\frac{\alpha}{2}$$

$$2d - R\alpha = 1$$

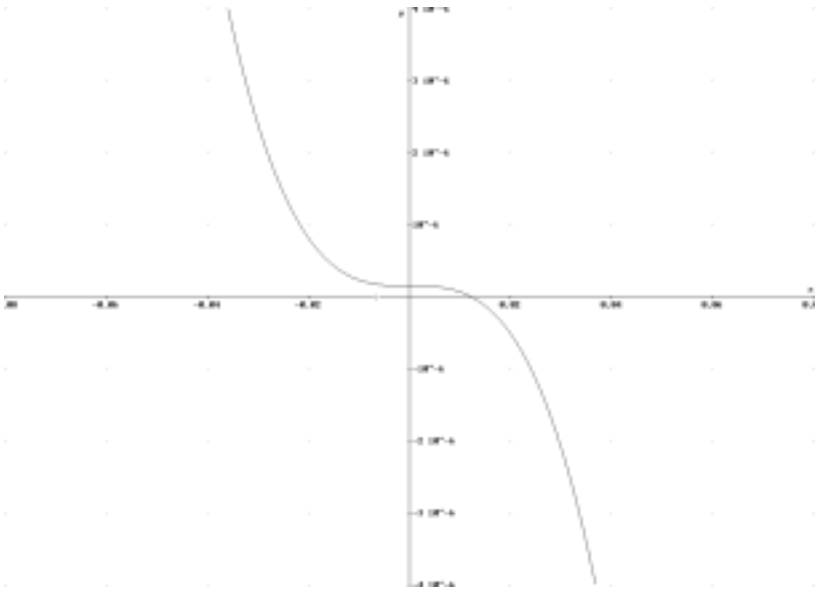
De les dos últimes equacions tenim:

$$\alpha = 2 \cdot \mathrm{tg}\frac{\alpha}{2} - \frac{1}{R}$$

Aquesta es una equació amb una sola incògnita α, equació que no sabem resoldre de forma analítica.

Com aquest, hi ha molts problemes que porten a equacions que no es poden resoldre analíticament. No obstant, els programes de càlcul com DERIVE o MATHEMATICA són una eina molt senzilla i eficaç per resoldre aquest tipus d'equacions.

Per exemple: fent la gràfica de la funció $y = x - 2\,\mathrm{tg}\frac{\alpha}{2} + \frac{1}{R}$



S'hi veu que hi ha una solució de l'equació. Aquesta es pot determinar numèricament amb qualsevol programa:

x = 0.0123738 radians, per tant **d** = $3.93924 \cdot 10^{4}$ m i **h**=121.85 m. **Resultat sorprenent !**

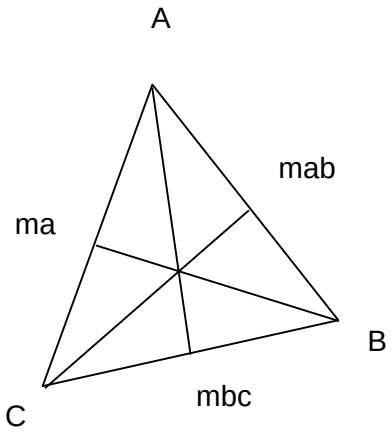
Exemple 2: Veiem un exemple en el que fem servir el programa de càlcul simbòlic *DERIVE* per demostrar que les coordenades del baricentre d'un triangle són les mitjanes aritmètiques de les coordenades dels seus tres vèrtex:

Considerem els punts A, B i C:

$\alpha := [\alpha 1, \alpha 2]$

$\beta := [\beta 1, \beta 2]$

$\chi := [\chi 1, \chi 2]$



els punts mitjans dels segments del triangle seran:

$$\mu\alpha\beta := \frac{\alpha + \beta}{2}$$

$$f\left(\frac{\alpha 1 + \beta 1}{2}, \frac{\alpha 2 + \beta 2}{2}\right)$$

$$\mu\alpha\chi := \frac{\alpha + \chi}{2}$$

$$f\left(\frac{\alpha 1 + \chi 1}{2}, \frac{\alpha 2 + \chi 2}{2}\right)$$

$$\mu\beta\chi := \frac{\beta + \chi}{2}$$

$$f\left(\frac{\beta 1 + \chi 1}{2}, \frac{\beta 2 + \chi 2}{2}\right)$$

Ensenyem a DERIVE la forma de trobar la recta que passa per dos punts:

$$PEXTA(\alpha, \beta) := \begin{pmatrix} \beta_2 - \alpha_2 \\ \beta_1 - \alpha_1 \end{pmatrix} \bullet (\xi - \alpha_1) + \alpha_2$$

Busquem la intersecció de les medianes que passen per a i mbc i per b i mac.

SOLVE(RECTA(a, mbc) = RECTA(b, mac), x)

$$f\xi = f\left(\frac{\alpha 1 + \beta 1 + \chi 1}{3}\right)$$

$$\xi := \frac{\alpha 1 + \beta 1 + \chi 1}{3}$$

RECTA(a, mbc)

$$\frac{\alpha 2 + \beta 2 + \chi 2}{3}$$

Trobem, per tant, que les coordenades del baricentre o punt d'intersecció de les medianes són:

$$f\left(\frac{\alpha 1 + \beta 1 + \chi 1}{3}, f\left(\frac{\alpha 2 + \beta 2 + \chi 2}{3}\right)\right)$$

Exemple 3: Com a il·lustració de com es pot aplicar la simulació per comprendre comportaments complexos comentaré el senzill experiment virtual que va proposar Christopher Langton en la Universitat de Michigan:

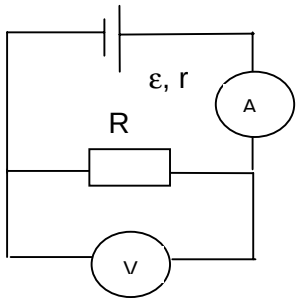
Va crear a l'ordinador una "formiga virtual", assignant-li una sèrie de regles senzilles per determinar la forma en què aquesta respondrà al seu entorn (una graella composta per cel·les).

- Si la formiga penetra en una cel·la blanca, continua desplaçant-se en la mateixa direcció que portava.
- Si la formiga s'introdueix en una cel·la blava, aquesta adoptarà el color groc i la formiga girarà a l'esquerra.
- Si la formiga s'introdueix en una cel·la groga, aquesta adoptarà el color blau i la formiga girarà a la dreta.

Va deixar simultàniament varies formigues en la mateixa graella i el seu comportament va resultar molt semblant al de les formigues reals: en primer lloc, van caminar sense rumb definit durant els primers instants i, a continuació, es van trobar unes a les altres i van formar una fila. L'aspecte més interessant d'aquest experiment consisteix en que Langton no havia programat explícitament a les formigues per a que presentaren un comportament social. Aquest comportament és el que es diu comportament "emergent".

Aquest exemple, el qual pot donar peu a força comentaris, recull molt bé la filosofia de la Simulació. Només s'han programat elements molts senzills del model de "formiga" i hem comprovat que aquests elements són els responsables del comportament emergent del conjunt de formigues. En el cas més conegut de simular la trajectòria d'un planeta que gira al voltant del sol, no tindria massa sentit imposar que el planeta segueixi una trajectòria el·líptica sinó que, el que s'ha d'imposar és la regla més senzilla que consisteix en el tipus de força que actua entre el planeta i el Sol i que ve determinat per la llei de Newton. Que la trajectòria sigui el·líptica ha de ser un dels resultats de la simulació.

Exemple 4: Per últim, presentaré un exemple d'uns dels aspectes més coneguts de l'ús de l'ordinador en Ciències: *el tractament i anàlisi de dades*. Aquest exemple consisteix a trobar la resistència interna d'un generador de corrent continu. Per això, hem construït un circuit com el de la figura:



La f.e.m del generador és $\varepsilon = 4.5$ V i la resistència externa R és 100 Ω.
La equació del circuito és:

$$V = \varepsilon - r\,I$$

La taula obtinguda per a diferents valors de la resistència R és:

Intensitat (A)	Dif. De potencial (V)
0.35	3.98
1.22	3.13
1.80	2.45
2.40	1.55
2.90	0.95
3.71	0.18

Amb els mètodes de regressió tradicionals podem trobar la recta de regressió. El pendent d'aquesta recta corres-pon a la resistència interna del generador. Amb qualse-vol eina de càlcul, com per exemple un full de càlcul Podem trobar el valor d'aquest pendent i el coeficient de Correlació que ens dóna la bondat d'aquest ajust.

0,35	3,98
1,22	3,13
1,8	2,45
2,4	1,55
2,9	0,95
3,71	0,18

PENDIENTE(B1:B6;A1:A6)	-1,17261127
COEF.DE.CORREL(A1:A6;B1:B6)	-0,99743905

Per tant, afirmar que la resistència interna del generador és de 1.17 Ω.

No exposaré aquí, encara que també és interessant, la forma de calcular l'error d'aquesta estimació.

Annex II. Codi del programa *Simula*

simula.c

```
#include "simula.h"
#include "forca.h"
#include "comprova.h"

int main(){
    int ct,par=1;
    struct particula p, p_ant;

    algorisme[0]=euler;
    algorisme[1]=euler_ric;
    algorisme[2]=verlet;

    //******************inici funcions gràfiques***************
    //*************************************************************
    allegro_init();
    install_keyboard();
    install_timer();
    set_gfx_mode(GFX_AUTODETECT,800,600,0,0);
    inici();
    set_pallete(desktop_pallete);
    //*************************************************************

    // lectura de dades del fitxer
    lectura_dades_generals();
    //inicialitzaci¢ de les dades de la part¡cula
    p.x=px0;p.y=py0;p.vx=pvx0;p.vy=pvy0;p.massa=pmassa;p.radi=pradi;

    dibuixar();
    parada(0,&p);
    for(ct=1;ct<=nombre_passos;ct++){
        p_ant=p;                    //guarda la configuració anterior
        algorisme[algint](&p,t);
                if(!traj)               circlefill(screen,fx(p_ant.x),fy(p_ant.y),p_ant.radi,color_fons);

        circlefill(screen,fx(p.x),fy(p.y),p.radi,color_part);

        rest(retard);

        textprintf(screen,font,601,100,4,"pos = (%.2lf,%.2lf)",p.x,p.y);
        textprintf(screen,font,601,110,4,"vel = (%.2lf,%.2lf)",p.vx,p.vy);
        textprintf(screen,font,601,120,5,"pas=%d temps = %.2lf",ct,t*ct);

        if (comprova(p,ct)) par=parada(&ct,&p);
        if (key[KEY_ESC]) par=parada(&ct,&p);
        if (pause==1) if (!(ct%passos_pause)) par=parada(&ct,&p);
        if (!par) break;    //si par=0 surt del bucle
        textprintf(screen,font, 601,400, 5, "                    ");
        textprintf(screen,font, 601,410, 5, "                  ");
        textprintf(screen,font, 601,420, 5, "                     ");
        textprintf(screen,font, 601,430, 5, "                      ");
        textprintf(screen,font, 601,440, 5, "                   ");
        textprintf(screen,font, 601,450, 5, "                              ");


    }
    textprintf(screen,font,601,300,5,"fi de la iteracio");

}



//*******Funcions que passen de coordenades lògiques a físiques**
//*****************************************************************
int fx(double lx){
   return (int)((lx-lox)/(lfx-lox)*(ffx-fox)+1);
}
```

```
//*****************************************************************
int fy(double ly){
   return (int)((ly-loy)/(lfy-loy)*(ffy-foy)+1);
}


//*****************************************************************
//**********Algorismes d'integració*****************************
//*****************************************************************
void euler(struct particula *p,double t){
    double ax;
    double ay;
    forca(p,&ax,&ay);          //calcula l'acceleració

    p->x=p->x+p->vx*t+0.5*ax*t*t;
    p->y=p->y+p->vy*t+0.5*ay*t*t;
    p->vx=p->vx+ax*t;
    p->vy=p->vy+ay*t;
}
//*****************************************************************

void euler_ric(struct particula *p,double t){
    struct particula pm;
    double ax;
    double ay;

    pm.massa=p->massa;
    forca(p,&ax,&ay);          //calcula l'acceleracio

    pm.vx=p->vx+0.5*ax*t;
    pm.vy=p->vy+0.5*ay*t;

    pm.x=p->x+0.5*p->vx*t;
    pm.y=p->y+0.5*p->vy*t;

    forca(&pm,&ax,&ay);

    p->x=p->x+pm.vx*t;
    p->y=p->y+pm.vy*t;

    p->vx=p->vx+ax*t;
    p->vy=p->vy+ay*t;

}
//*****************************************************************

void verlet(struct particula *p, double t){
    //aquest algorisme només funcionarà si
    //la força no depén de la velocitat

    double ax1,ay1,ax2,ay2;

    forca(p,&ax1,&ay1);
    p->x=p->x+p->vx*t+0.5*ax1*t*t;
    p->y=p->y+p->vy*t+0.5*ay1*t*t;

    forca(p,&ax2,&ay2);
    p->vx=p->vx+0.5*(ax1+ax2)*t;
    p->vy=p->vy+0.5*(ay1+ay2)*t;
}


//*****************************************************************
//**********        Dibuixa       ****************************
//*****************************************************************

void dibuixar(){
    clear(screen);
    text_mode(0);
    rect(screen,fox-1,foy-1,ffx+1,ffy+1,color_vora);
    rectfill(screen,fox,foy,ffx,ffy,color_fons);
    line(screen,fx(lox),fy(0),fx(lfx),fy(0),0);
    line(screen,fx(0),fy(loy),fx(0),fy(lfy),0);
```

```
    textprintf(screen,font,fox,foy,4,"(%.2lf,%.2lf)",lox,loy);
    textprintf(screen,font,ffx-118,ffy-8,4,"(%.2lf,%.2lf)",lfx,lfy);
    textprintf(screen,font,601,10,7,"DADES DE LA PARTICULA");
    textprintf(screen,font,601,20,4,"(x0,y0)  =(%.2lf,%.2lf)",px0,py0);
    textprintf(screen,font,601,30,4,"(vx0,vy0)=(%.2lf,%.2lf)",pvx0,pvy0);
    textprintf(screen,font,601,40,4,"massa     =%.2lf",pmassa);
    textprintf(screen,font,601,60,5,"inc de temps   =%.2lf",t);
    textprintf(screen,font,601,70,5,"nombre
de passos=%d",nombre_passos);

    textprintf(screen,font,601,80,5,"temps total
   =%.2lf",t*nombre_passos);
    circlefill(screen,fx(px0),fy(py0),pradi,color_part);
}


//*****************************************************************
//**********        Parada        ****************************
//*****************************************************************

int parada(int *ct, struct particula *p){


      //si aquesta funció retorna un 1 continuarà la simulació
      //si retorna un 0 acabarà la simulació
      int tecla;

      textprintf(screen,font, 601,400, 5, "c(desar i continuar)");
      textprintf(screen,font, 601,410, 5, "d(desar i sortir)");
      textprintf(screen,font, 601,420, 5, "s(sortir sense desar)");
      textprintf(screen,font, 601,430, 5, "x(canviar configuració)");
      textprintf(screen,font, 601,440, 5, "n(comencar de nou)");
      textprintf(screen,font, 601,450, 5, "altra (continuar sense sortir)");

      menu:
      tecla=readkey();
      if((tecla&0xff)=='c') {desar(*ct,*p);return 1;}
      if((tecla&0xff)=='d') {desar(*ct,*p);return 0;}
            if((tecla&0xff)=='s') return 0;
      if((tecla&0xff)=='x') {canviar_configuracio();goto menu;}
      if((tecla&0xff)=='n') {*ct=0;
                        dibuixar();
                        p->x=px0;p->y=py0;p->vx=pvx0;p->vy=pvy0;
                        p->massa=pmassa;p->radi=pradi;
                        return 1;}

      return 1;
}


//*****************************************************************
//**********        Desar         ****************************
//*****************************************************************
int desar(int ct, struct particula p){
      FILE *fp;

      if((fp=fopen(arxiu_sortida,"a"))==NULL){
        printf("error, no es pot obrir l'arxiu\n");
        return;
      }

      if(!ftell(fp))          //Si l'arxiu no s'ha obert abans
        fprintf(fp,"ct, t, p.x, p.y, p.vx, p.vy\n");

      fprintf(fp,"%d, %lf, %lf, %lf, %lf, %lf\n",ct,ct*t,p.x,p.y,p.vx,p.vy);
      fclose(fp);
}


//*****************************************************************
//**********        Canviar configuracio   *********************
//*****************************************************************
```

```
int canviar_configuracio(){

      char num[]="          ";
      int n;
            BITMAP *copia_pantalla=create_bitmap(SCREEN_W,SCREEN_H);
            blit(screen, copia_pantalla, 0,0,0,0,SCREEN_W,SCREEN_H);
      allegro_exit();

     do{
      clrscr();
      printf("            CANVI DE CONFIGURACIO");


      gotoxy(10, 3);printf("1. lox=%lf",lox);
      gotoxy(10, 4);printf("2. loy=%lf",loy);
      gotoxy(10, 5);printf("3. lfx=%lf",lfx);
      gotoxy(10, 6);printf("4. lfy=%lf",lfy);
      gotoxy(10, 7);printf("5. t=%lf",t);
      gotoxy(10, 8);printf("6. nombre de passos=%d",nombre_passos);
      gotoxy(10, 9);printf("7. pause=%d",pause);
      gotoxy(10, 10);printf("8. passos pause=%d",passos_pause);
      gotoxy(10, 11);printf("9. retard=%d",retard);
      gotoxy(10, 12);printf("10. traj=%d",traj);
      gotoxy(10, 13);printf("11. algorisme d'integracio=%d",algint);
      gotoxy(10, 14);printf("12. control=%d",control);
      gotoxy(10, 15);printf("13. px0=%lf",px0);
      gotoxy(10, 16);printf("14. py0=%lf",py0);
      gotoxy(10, 17);printf("15. pvx0=%lf",pvx0);
      gotoxy(10, 18);printf("16. pvy0=%lf",pvy0);
      gotoxy(10, 19);printf("17. pmassa=%lf",pmassa);
      gotoxy(10, 20);printf("18. pradi=%d",pradi);

      gotoxy(10,22);printf("voleu modificar alguna dada (s/n)");
      n=getch();
      if(n=='s'||n=='S') {
           gotoxy(10,23);printf("dada a modificar: ");scanf("%s",&num);
           n=atoi(num);
           if((n>=1)&&(n<=18)){
              gotoxy(40,n+2);scanf("%s",&num);
              switch (n){
                    case 1:lox=atof(num);break;
                    case 2:loy=atof(num);break;
                    case 3:lfx=atof(num);break;
                    case 4:lfy=atof(num);break;
                    case 5:t=atof(num);break;
                    case 6:nombre_passos=atoi(num);break;
                    case 7:pause=atoi(num);break;
                    case 8:passos_pause=atoi(num);break;
                    case 9:retard=atoi(num);break;
                    case 10:traj=atoi(num);break;
                    case 11:algint=atoi(num);break;
                    case 12:control=atoi(num);break;
                    case 13:px0=atof(num);break;
                    case 14:py0=atof(num);break;
                    case 15:pvx0=atof(num);break;
                    case 16:pvy0=atof(num);break;
                    case 17:pmassa=atof(num);break;
                    case 18:pradi=atoi(num);break;
              }
           }
      }
     }while(n>0&&n<19);
      allegro_init();
      install_keyboard();
      install_timer();
      set_gfx_mode(GFX_AUTODETECT,800,600,0,0);
      set_pallete(desktop_pallete);


      set_gfx_mode(GFX_AUTODETECT,800,600,0,0);
      set_pallete(desktop_pallete);
      blit(copia_pantalla,screen, 0,0,0,0,SCREEN_W,SCREEN_H);
}
```

```
void lectura_dades_generals(){
        FILE *fp;
        char temp[80];       //aquesta variable nom,s es fa servir
                             //per llegir una cadena que no es far… servir

       if((fp=fopen(arxiu_entrada,"r"))==NULL){
         textprintf(screen,font,10,10,7,"error, no es pot obrir l'arxiu");
         readkey();
         return;
       }

       //coordenades f¡siques
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%d %d %d %d ",&fox,&foy,&ffx,&ffy);

       //coordenades l•giques
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%lf %lf %lf %lf ",&lox,&loy,&lfx,&lfy);

       //colors
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%d %d %d ",&color_fons,&color_vora,&color_part);

       //temps i control
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%lf %d %d %d ",&t,&nombre_passos,&pause,&passos_pause);
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%d %d %d %d ",&retard,&traj,&algint,&control);

       //condicions inicials
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%lf %lf %lf %lf ",&px0,&py0,&pvx0,&pvy0);
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%lf %d ",&pmassa,&pradi);

       //nom de l'arxiu de dades
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%s ",arxiu_sortida);

       //variables addicionals
       fscanf(fp,"%[^\n]",temp);
       fscanf(fp,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf ",
          &var0,&var1,&var2,&var3,&var4,&var5,&var6,&var7,&var8,&var9);

       fclose(fp);
}


//**************************************************************************
//           INICI
//**************************************************************************

void inici(){

  BITMAP *the_image;
  PALLETE the_pallete;
  the_image = load_bitmap("logo.pcx", the_pallete);
  set_pallete(the_pallete);

  /* blit the image onto the screen */
  blit(the_image, screen, 0, 0, (SCREEN_W-the_image->w)/2, (SCREEN_H-the_image->h)/2, the_image->w, the_image-
>h);

  /* destroy the bitmap */
  destroy_bitmap(the_image);
  readkey();
 }
```

comprova.h

```
//*****************************************************************
//**********        Comprovació   ********************************
//*****************************************************************
//*****************************************************************
int comprova(struct particula p,int ct){
//     if (fabs(p.y)<0.01) return 1;
     return 0;

}
//*****************************************************************
```

simula.h

```
#include <string.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "allegro.h"

//*****************************************************************
//                      VARIABLES
//*****************************************************************
//coordenades fisiques
unsigned int    fox=1,   foy=1,  ffx=598,  ffy=598;

//coordenades logiques
double          lox=-100,   loy=20,  lfx=2,    lfy=-20;

//colors
unsigned int     color_fons=4,          color_vora=1;
unsigned int     color_part=67;

//temps i control
double t=0.001;
unsigned int     nombre_passos=1000000;
int   pause=0,   passos_pause=100;
int   retard=0,  traj=1;
unsigned int    algint=1,       control=0;

//condicions inicials
double px0=0,   py0=0,   pvx0=1,   pvy0=0,   pmassa=1  ;int pradi=1;

char arxiu_entrada[]="in.txt",arxiu_sortida[]="out.txt";


//variables addidcionals per la forca
double var0,var1,var2,var3,var4,var5,var6,var7,var8,var9;

//**************definició de l'estructura particula************************

struct particula{
       double x,y;              //posició
       double vx,vy;             //velocitat
       double massa;
       int radi;                //radi del dibuix en pixels (no real)
};
```

```
//*****************************************************************
//                      FUNCIONS
//*****************************************************************

void euler(struct particula *,double);
void euler_ric(struct particula *,double);
void verlet(struct particula *, double);
void (*algorisme[4])(struct particula *,double); //punter a algorisme

int fx(double );                  //calcula les coordenades físiques a par-
int fy(double );                  //tir de les coordenades lògiques.
int comprova(struct particula, int ct);
void forca(struct particula *p, double *ax, double *ay);
void dibuixar();
int parada(int*, struct particula*);
int desar(int, struct particula);
int canviar_configuracio();
void lectura_dades_generals();
void inici();
```

Forca.h

```
//*****************************************************************
//**********        For‡a        ******************************
//*****************************************************************

void forca (struct particula *p, double *ax, double *ay){

            double alfa;              //alfa=G*M
            double r3;

      alfa=var0;
      r3=sqrt(pow(p->x*p->x+p->y*p->y,3));


            *ax=-alfa*p->x/r3;
            *ay=-alfa*p->y/r3;
}
```

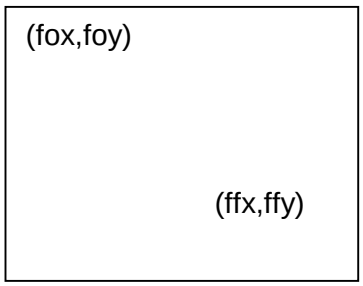
Annex III. Descripció detallada de les variables i funcions de *Simula*

Variables del programa:

Coordenades físiques:

```
unsigned int fox=1, foy=1, ffx=598, ffy=598;
```

Corresponen a les coordenades del punt superior esquerra (*fox, foy*) i del punt inferior dreta (*ffx,ffy*) de la zona de la pantalla on es visualitzarà l'animació. S'ha de tenir en compte que si es fa servir la resolució 800x600, aquests dos números representen cotes superiors de les coordenades físiques de punts de la pantalla. Els valors d'aquestes coordenades ha de ser, evidentment, enters positius.



Coordenades lògiques:

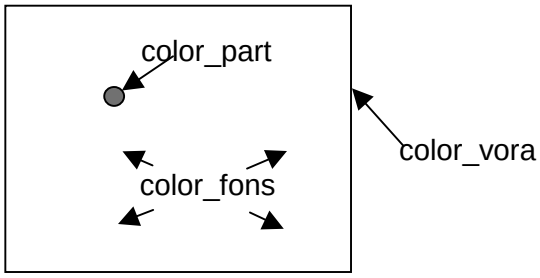
```
double lox=-100, loy=20, lfx=2, lfy=-20;
```

Corresponen a les coordenades lògiques que associem als punts superior esquerra i inferior dreta de zona d'animació. El canvi d'aquestes variables representa únicament la visualització a una escala diferent de l'animació, no canvia la zona de la pantalla.

Colors:

```
unsigned int color_fons=4, color_vora=1, color_part=67;
```

Aquestes tres variables controlen respectivament els colors de la zona d'animació, del rectangle que envolta a la zona d'animació i de la partícula.



Variables de control de temps:

```
double t=0.001;
unsigned int nombre_passos=10000;
```

```
int pause=0,        passos_pause=100,    retard=0;
```

La variable *t* representa el pas de temps. Un valor petit de *t* millora la precisió però fa l'animació més lenta.

La variable *nombre_passos* és el total de passos de temps abans d'acabar definitivament l'animació.

La variable *pause* és una variable que pot contenir els valors 0 i 1. Un valor 1 fa que es faci una pausa cada *passos_pause* passos de temps.

La variable *retard* pot forçar un retard de temps per tal de controlar la velocitat de l'animació. Un valor de 0 representa cap retard i, per tant, la màxima velocitat possible que permet l'algorisme i el pas de temps.

Variable de trajectòria:

```
int traj=1;
```

La variable *traj* pot contenir els valors 0 i 1. Un valor 0 fa que en cada pas de temps esborri de la pantalla la partícula i la torni a dibuixar en la nova posició. Un valor 1 fa que no s'esborri la partícula i, per tant, es veurà el rastre de la trajectòria.

Algorisme d'integració:

```
unsigned int algint=1;
```

Es pot seleccionar un de tres algorismes d'integració: L'algorisme d'Euler, l'algorisme d'Euler-Richarson i l'algorisme de Verlet.

algint=0 correspon a l'algorisme d'Euler
algint=1 correspon a l'algorisme d'Euler Richarson
algint=2 correspon a l'algorisme de Verlet

Condicions inicials de la partícula:

```
double px0=0, py0=0, pvx0=1, pvy0=0, pmassa=1;
int pradi=1;
```

Aquestes variables es fan servir per determinar la posició i la velocitat de la partícula, així com la massa i el radi. La massa és una dada dinàmica que afecta a la trajectòria de la partícula. El radi només afecta a la forma de visualització de la partícula.

Nom dels arxius d'entrada i sortida:

```
char arxiu_entrada, arxiu_sortida;
```

Aquestes variables contenen el nom dels arxius d'entrada de dades i sortida de dades.

Funcions del programa:

Algorismes d'integració:

```
void euler(struct particula *, double);
void euler_ric (struct particula *, double);
void verlet(struct particula *, double);
void (*algorisme[4])( struct particula *, double);
```

Les tres primeres funcions contenen els respectius algorismes d'integració. Tenen dos arguments: la estructura partícula (que conté les dades dinàmiques d'aquesta, és a dir, posició, velocitat i massa) i un segon argument que és el pas de temps. Cap d'aquestes funcions no torna valors.

algorisme és un punter a qualsevol d'aquestes funcions.

Funcions de conversió de coordenades físiques a partir de les coordenades lògiques:

```
int fx(double);
int fy(double);
```

Els respectius algorismes calculen unes coordenades lògiques. Per tal de representar la posició de la partícula a la pantalla s'ha de trobar les coordenades físiques on realment es dibuixarà la partícula.

Lectura i escriptura de dades:

```
void lectura_dades_generals();
void desar(int*,struct particula*);
```

La funció *lectura_dades_generals()* llegeix les dades de la simulació de l'arxiu indicat per la variable *arxiu_entrada*.

La funció *desar(int*,struct particula*)* permet desar en l'arxiu indicat per la variable *arxiu_sortida* les dades instantànies de la partícula per una anàlisi posterior.

Model de força:

```
void forca(struct particula *p, double *ax, double *ay);
```

Aquesta és la funció que conté la informació del model concret que s'està simulant. Hi haurà una funció forca per a cada tipus de simulació, per aquesta raó, el contingut d'aquesta funció es troba fóra del fitxer *simula.c*.

Aquesta funció calcula l'acceleració instantània a partir de les dades dinàmiques de la partícula i és cridada per les funcions que fan l'algorisme d'integració.

Altres funcions:

void dibuixar();

Aquesta funció dibuixa la zona d'animació i escriu el valor de les condicions inicials per pantalla.

*int parada(int *,struct particula *);*

Aquesta funció se crida si es dóna alguna de les diferents condicions que atura la simulació:

- Prèmer la tecla ESC
- El pas de temps és múltiple de *passos_pause* i la variable *pause* conté el valor 1.
- Si es dóna alguna condició prevista a la funció *comprova*.

Aquesta funció fa que es mostri el següent menú a la pantalla:

- desar i continuar
- desar i sortir
- sortir sense desar
- canviar configuració
- començar de nou
- continuar sense sortir

void canviar_configuració();

Aquesta funció es crida si es selecciona la quarta opció del menú anterior i fa aparèixer la llista de totes les variables que controlen la simulació i les permet modificar.

void comprova(struct particula, int);

Aquesta funció definida fóra de l'arxiu *simula.h* permet definir condicions de temps o condicions referits a la posició o velocitat de la partícula per tal d'aturar la posició. Per exemple es pot donar la condició de que s'aturi quan la velocitat sigui 0, o que la posició sigui més gran que 2, etc.

Annex IV. 1ª Pre-Olimpíada Informàtica Catalana. Aspectes generals i Calendari

Nom del concurs:	**I Pre-Olimpíada Informàtica Catalana**
Edat o nivell dels participants:	**ESO i primer curs de batxillerat o cicles formatius de grau mitjà**. (alumnes que no hagin complert 18 anys abans del 31 de desembre de 2001) No poden participar els de segon curs de batxillerat o cicles formatius.
Nombre de fases:	**Dos**: una fase prèvia a distància i una segona fase presencial en una o més seus.
Nivell de dificultat del concurs:	És objectiu d'aquest concurs que el nivell general de les proves sigui sensiblement **inferior** que els de les olimpíades informàtiques tradicionals. Aquesta mesura pretén fomentar la participació dels alumnes. Els exercicis proposats es classificaran en tres nivells diferents: **primer nivell**: Problemes gairebé elementals que tothom amb uns coneixements mínims pugui desenvolupar sense problemes. **segon nivell**: Problemes que necessitin un plantejament previ més elaborat. **tercer nivell:** Problemes que obliguin al concursant a desenvolupar estratègies de resolució més sofisticades.
Patrocini i Organització:	**Subdirecció General de Tecnologies de la Informació**
Comitè organitzador:	*Jordi Castells* *Santiago Manrique* *Miquel Gomila* *Francisco J. Ruiz*
Jurat:	*Jordi Castells* *Santiago Manrique* *Francisco J. Ruiz* actuant aquest últim com a president.
Llenguatges admesos	C/C++, Pascal, Visual BASIC

CALENDARI

DATA	ACCIÓ	DESCRIPCIÓ
26 febrer	comença la	Es fa mitjançant formulari electrònic. S'ha de fer

	inscripció	constar les dades del concursant i les dades d'un professor que l'avali. Per cada inscripció feta s'enviarà un missatge de correu electrònic amb la confirmació de la inscripció i es donarà un codi de tres xifres que identificarà al concursant.
15 març	últim dia d'inscripció	
22 març	comença la fase prèvia	Es lliuraran els enunciats de tres exercicis a l'adreça electrònica de contacte.
28 març	últim dia per lliurar els exercicis de la fase prèvia	S'ha d'enviar per correu electrònic dos fitxers per problema resolt: un amb el codi font i un altre amb l'executable.
6 abril	publicació del seleccionats	Es publicarà el nom dels seleccionats a la pàgina oficial i se'ls enviarà un correu electrònic per notificar-lo. En el cas de que hagi més d'una seu es comunicarà la llista de participants per seus.
21 abril	fase final	

FEBRER 2001

dl	dm	dx	dj	dv	ds	dm
			1	2	3	4
5	6	7	8	8	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	**27**	**28**				

MARÇ 2001

dl	dm	dx	dj	dv	ds	dm
			1	**2**	**3**	**4**
5	**6**	**7**	**8**	**9**	**10**	**11**
12	**13**	**14**	**15**	16	17	18
19	20	21	**22**	**23**	**24**	**25**
26	**27**	**28**	29	30	31	

ABRIL 2001

dl	dm	dx	dj	dv	ds	dm
						1
2	3	4	5	**6**	7	8
9	10	11	12	13	14	15
16	17	18	19	20	**21**	22
23	24	25	26	27	28	29
30						

Annex V. 1ª Pre-Olimpíada Informàtica Catalana. Bases

- Aquest concurs està destinat als alumnes d'ESO, 1r de batxillerat i 1r curs de cicles formatius de grau mitjà dels centres de Catalunya que no hagin fet 18 anys abans del dia 31 de desembre de 2001.

- La competició té caràcter individual. Per inscriure's, el concursant haurà d'omplir el formulari que trobarà en aquesta pàgina i enviar-lo abans del **15 de març de 2001**.

- Tota inscripció ha d'estar avalada per un professor del centre on l'alumne cursi els seus estudis.

- La inscripció a aquest concurs és **gratuïta**. L'Organització **no** correrà amb les possibles despeses de desplaçament dels concursants.

- El dia **22 de març** es començarà la fase prèvia del concurs. Es lliurarà, mitjançant correu electrònic als alumnes inscrits, un conjunt de 3 exercicis. L'alumne haurà de resoldre sense ajuda els exercicis i trametre'ls abans de les **18 hores** del dia **28 de març**.

- El jurat examinarà els exercicis lliurats i triarà els millors treballs per participar a la fase final.

- La llista dels seleccionats per participar a la fase final serà publicada a la pàgina oficial de la Pre-Olimpíada i comunicada per correu electrònic als interessats abans del dia **6 d'abril**.

- La fase final es realitzarà el dissabte **21 d'abril** en una o diverses seus, depenent del nombre de concursants seleccionats i de la seva distribució geogràfica.

- Els concursants de la fase final hauran d'acreditar documentalment, el mateix dia de la prova, el compliment dels requisits del punt 1 d'aquestes bases.

- A la fase final, els concursants hauran de resoldre un conjunt de 5 exercicis de programació de dificultat creixent. Per aquesta tasca disposaran d'un total de 4 hores.

- Els exercicis, tant de la fase prèvia com de la fase final, consistiran en la confecció de programes que llegeixen dades d'un fitxer d'entrada i escriguin la solució en un fitxer de text amb un format perfectament especificat a l'enunciat. Els fitxers, tant d'entrada com de sortida, seran totalment independents de la plataforma. No es faran servir mai les capacitats gràfiques dels llenguatges de programació.

- En la fase final, i per la resolució dels exercicis, no es podrà disposar més que dels estris d'escriptura i l'ordinador subministrats per l'Organització. Aquest ordinador disposarà de tots els compiladors dels llenguatges permesos al concurs. Els compiladors es carregaran amb totes les ajudes que tinguin.

- En les 4 hores de duració de la prova, el concursant no podrà comentar els exercicis amb cap altre concursant ni fer servir llibres, apunts, telèfons mòbils o altre tipus de dispositius de comunicació.

- Els llenguatges de programació permesos són: **C/C++**, **Pascal** i **Basic (VisualBasic)**. En tots els casos, tant a la fase prèvia com a la fase final, el concursant haurà de lliurar el fitxer amb el codi font en format ASCII (.c, .cpp, .pas, .bas) i el fitxer compilat (.exe).

- Tots els exercicis de la prova tindran una mateixa puntuació de **10 punts**.

- La correcció de la prova es farà automàticament amb uns fitxers d'entrada de prova. El jurat podrà adjudicar la totalitat o part de la puntuació d'un exercici segons el nombre i tipus de fitxers prova que superi. En cas de que la correcció automàtica no conclogui una classificació, el jurat podrà perfilar-la en funció d'aspectes relacionats amb el codi escrit pel concursant: optimització, temps d'execució, documentació, etc.

- La composició del jurat es farà pública juntament amb la classificació final dels concursants.

- Els primers classificats rebran premis en material informàtic.

- La decisió del jurat serà inapel·lable.

Annex VI. 1ª Pre-Olimpíada Informàtica Catalana. Informació sobre compiladors

Els llenguatges oficials del concurs seran C/C++, Pascal i Basic (Visual Basic). Els exercicis resolts es podran enviar en qualsevol d'aquests llenguatges, sigui quina sigui la versió, no obstant, els compiladors i versions de referència i que es faran servir a la fase final són:

- PASCAL : Free Pascal versió 1.0.4 per a Windows
- C/C++ : DJGPP versió 2.95.2
- BASIC: Visual BASIC, versió 6

Free Pascal

Aquest compilador gratuït és pràcticament 100% compatible amb el Turbo Pascal 7.0. També inclou una interfície de desenvolupament (IDE) semblant al del compilador de Borland. Es pot baixar directament des de l'adreça WEB: http://www.freepascal.org.

DJGPP

Aquest és un compilador C/C++ de 32 bits per a PC, també gratuït, que es pot aconseguir juntament amb les instruccions d'instal·lació i d'altres eines, com un entorn de desenvolupament de projectes, un depurador i llibreries gràfiques a http://www.delorie.com/djgpp.

Visual Basic

Aquest és una de les parts del paquet de Microsoft Visual Studio que està sent subministrada pel Departament d'Ensenyament.

Encara que aquesta eina de desenvolupament es fa servir fonamentalment per a aplicacions gràfiques, no es farà servir cap d'aquestes propietats per resoldre els exercicis.

Per crear una aplicació que s'ajusti als requeriments d'aquest concurs, és a dir, que al executar-lo:

- agafi les dades del fitxer d'entrada automàticament,
- faci automàticament els càlculs,
- escrigui la solució en en fitxer de sortida i
- finalitzi l'execució,

s'ha d'escriure el codi dintre d'un bloc *sub ... end sub* de l'apartat general. Després, per tal que aquest codi s'executi automàticament, s'ha de cridar a aquesta subrutina des de l'event *Sub Form_Load()...End Sub*, posant després *End* perquè el programa acabi automàticament. Per exemple:

```
Sub daus()
(...)
End Sub
Sub Form_Load()
daus
End
End Sub
```

A l'apartat d'exercicis d'entrenament hi ha un exemple resolt amb el Visual Basic

Annex VII. 1ª Pre-Olimpíada Informàtica Catalana. Exercicis d'Entrenament

A l'apartat d'Exercicis d'Entrenament es pot trobar aquesta col·lecció d'exercicis per tal que el futur concursant es faci una idea del tipus d'exercicis i els formats dels arxius d'entrada i sortida. Aquests exercicis estan classificats en tres tipus de dificultat, fàcil (1), menys fàcil (2), i difícil (3). Com exemple hi podem trobar la solució a tres d'ells en els tres llenguatges admesos en la competició: Pascal, C i Visual BASIC:

Dif	títol de l'exercici	solució
1	La multiplicació	
1	Les paraules al mirall	
1	Algoritme d'Euclides	euclid.pas
2	La divisió	divisio.c
2	Classificació dels nombres decimals	
2	Llançament de n daus	daus.frm
2	Resultat d'un partit	
2	Sopa de Lletres	
2	Algoritme 3n+1	
2	Justificació d'un text	
3	Laberint	
3	Màxim rectangle	

1. La multiplicació. Dificultat 1

Aquest programa pot ser útil per la correcció de multiplicacions. El programa ha d'efectuar el producte de dos nombres enters de **n** xifres i de **m** xifres respectivament, però no directament, sinó mostrant totes les multiplicacions parcials, és a dir, el fitxer de sortida contindrà **m+1** línies, les primeres **m** línies correspondran al producte de cadascuna de les **m** xifres del segon nombre pel primer nombre i la darrera línia contindrà el resultat total de la multiplicació.

Exemple:

12535 x 253 37605 62675 25070 3171355	**fitxer d'entrada:** 12535 253 **fitxer de sortida:** 37605 62675 25070 3171355

Ampliació: El fitxer de sortida podria contenir els espais necessaris per la correcta alineació de les xifres.

2. Les paraules al mirall. Dificultat 1

Aquest programa que es proposa és ja un clàssic en el conjunt d'exercicis elementals de programació. Es tracta d'invertir l'ordre de les lletres d'una paraula de **n** lletres.

Exemple:

	fitxer d'entrada: casa **fitxer de sortida:** asac

3. Algoritme d'Euclides. Dificultat 1

Euclides va ser un gran matemàtic i filòsof de l'antiguitat. Va idear un procediment per tal d'obtenir el **màxim comú divisor (MCD)** de dos nombres.
Donats dos nombres enters **A** i **B** es divideix el més gran entre el més petit. Si la resta **R** de la divisió es **0**, el divisor **B** és el MCD, en cas contrari, **B** es converteix en dividend i **R** en divisor. Tornem a fer la divisió, si la resta d'aquesta nova divisió es **0**, el divisor **R** és el MCD, en cas contrari, el divisor es converteix en dividend i la resta de la divisió en divisor, i tornem a fer la divisió. Fent això successivament trobarem alguna vegada resta **0**, en aquest moment, el divisor de la divisió serà el MCD.

El fitxer d'entrada contindrà els dos nombres enters i el fitxer de sortida contindrà el MCD.

Exemple:

MCD(160, 60) 160 entre 60 dóna resta 40 60 entre 40 dóna resta 20 40 entre 20 dóna resta 0, per tant 20 és el MCD	**fitxer d'entrada:** 160 60 **fitxer de sortida** 20

4. La divisió . Dificultat 2

No fa falta explicar en què consisteix l'algoritme per fer una divisió entre dos nombres enters o decimals. Pot ser, l'ús indiscriminat de calculadores ha fet que aquest conegut algoritme s'hagi oblidat una mica. El programa que et plantegem és un programa per reproduir l'algoritme tradicional de la divisió. El fitxer d'entrada contindrà dos nombres enters, un de **n** xifres (el dividend) i un altre de **m** xifres (el divisor). El fitxer de sortida haurà de contenir en primer lloc el quocient enter de la divisió i a continuació tots els nombres parcials de l'algoritme de la divisió.

Exemple:

53298 |25 32 2131 79 48 23	**fitxer d'entrada:** 53298 25 **fitxer de sortida:** 2131 32 79 48 23

S'hauria de tenir en compte què passa si el dividend és més petit que el divisor i també s'ha de tenir en compte que si el quocient conté la xifra zero, es baixen simultàniament dos o més xifres.

Ampliació: El programa podria permetre les dades decimals en lloc d'enteres. També es pot continuar la divisió (baixant zeros) fins trobar resta 0, fins un nombre màxim de decimals o fins trobar el període.

Solució:

```
#include <stdio.h>
#include <math.h>              /*pow()*/

int xifres(int a);             /*calcula el nombre de xifres d'a*/
int xifra(int a, int b);       /*calcula la xifra que ocupa la posici¢ b
                               en el nombre a*/

int main() {
    FILE *fint, *fout;
    int dividend, divisor;
```

```
    int xif;                /*nombre de xifres del divisor*/
    int re,ct;              /*variables auxiliars*/

 /* s'obren els fitxers d'entrada i de sortida*/
    if((fint=fopen("entrada.dat","r"))==NULL){
            printf("no es pot obrir el fitxer d'entrada\n");
            exit(1);
    }

     if((fout=fopen("sortida.dat","w"))==NULL){
            printf("no es pot obrir el fitxer de sortda\n");
            exit(1);
    }

 /* es llegeix les dades del fitxer d'entrada*/
    fscanf(fint,"%d%d",&dividend,&divisor);

 /* es calcula el nombre de xifres del divisor */
    xif=xifres(divisor);

 /* en el cas de que el dividend<divisor torna 0,dividend i acaba */
    if (dividend<divisor){
        fprintf(fout,"%d\n%d",0,dividend);
        exit(0);
    }

 /* escriu en primer lloc el quocient */
    fprintf (fout,"%d\n",dividend/divisor);

 /* saca les primeres xif xifres del dividend */
    re=(int)dividend/pow(10,xifres(dividend)-xif);

 /* procŠs iteratiu del c…lcul dels restos parcials */
    for(ct=xif+1;ct<=xifres(dividend);ct++){
            re=re%divisor*10+xifra(dividend,ct);
            if (re>=divisor) fprintf(fout,"%d\n",re);
    }

 /* per £ltim escriu el reste final de la divisi¢*/
    fprintf(fout,"%d\n",re%divisor);   /*escriu l'ultim reste*/
}



 /*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
 /* funcions auxiliars utilitzades*/

 /*nombre de xifres de a*/
int xifres(int a) {

       int n;
       for (n=1;n<10;n++){
           if (a<pow(10,n)) return n;
       }
       return(0);
}

 /* xifra b-,sima de a*/
int xifra(int a,int b){

       if(xifres(a)<b) {
           printf("error xifra\n");
           exit(1);
       }
       return ((int)(a/pow(10,xifres(a)-b)))%10;
}
```

5. Classificació dels nombres decimals. Dificultat 2

Com ja se sap, tot nombre racional, és a dir, aquell que és quocient de dos nombres enters, sempre es pot representar en forma de decimal finit o periòdic, per exemple, el nombre 2/5 és un nombre decimal finit perquè és igual a 0.4, però 4/7 és un nombre decimal periòdic amb un període format per sis xifres. És fàcil demostrar que una fracció es pot representar com un decimal finit si i només si el denominador de la fracció simplificada és de la forma $\mathbf{2^n \cdot 5^m}$. L'exercici consisteix en llegir d'un fitxer de text amb dos nombres enters i s'ha d'esbrinar si el nombre que representa la fracció és un nombre decimal finit o periòdic. Pot ser útil per aquest programa la realització abans del càlcul del MCD per la simplificació de la fracció que proposa el problema Algoritme d'Euclides.

Exemple:

$\frac{24}{15}=\frac{8}{5}$ el denominador és potència de 5, per tant, és finit.	**fitxer d'entrada**: 24 15 **fitxer de sortida:** finit

Ampliació: El 2 i el 5 apareixen en el programa pel fet que són els divisors de 10. Modifiqueu el programa per tal de que introduïda qualsevol fracció amb els numerador i denominador expressats en qualsevol base el programa pugui determinar si en aquesta base la fracció tindrà una forma decimal finita o no. S'ha de tenir en compte que un nombre que en una base es finit, en altra base podria no ser-ho.

6. Llançament de n daus. Dificultat 2

Trobeu totes les formes possibles de que amb **m** daus traiem un **n.** Per exemple: amb 2 daus hi ha 3 formes de sortir un 4: 13,22,31. S'ha de tenir en compte que **n** no pot ser més petit que **m** ni més gran que **6m**. El fitxer d'entrada contindrà dos nombres enters **n** i **m**, i el programa ha de tornar el nombre de formes d'obtenir la puntuació **m** important l'ordre dels daus. En el cas de que **m** no estigui a l'interval [**n , 6n]** haurà de tornar només un 0.

Exemple:

	fitxer d'entrada: 2 4 **fitxer de sortida:** 3

Solució:

```
VERSION 5.00
Begin VB.Form Form1
   Caption        =   "Form1"
   ClientHeight   =   3195
   ClientLeft     =   60
   ClientTop      =   345
   ClientWidth    =   4680
   LinkTopic      =   "Form1"
```

```
   ScaleHeight     =   3195
   ScaleWidth      =   4680
   StartUpPosition =   3  'Windows Default
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Function minim(n, m)
If n > m Then minim = m Else minim = n
End Function


Function f(n, m)
'funció recursiva que calcula el nombre de formes de treure
'la puntuació n amb m daus
Dim ct As Integer
Dim tem As Long


If m < 1 Then
   f = 0
   Exit Function
End If

If m = 1 Then
   If n < 1 Or n > 6 Then
      f = 0
      Exit Function
   Else
      f = 1
      Exit Function
   End If
Else

   For ct = 1 To minim(6, n - m + 2)
      temp = temp + f(n - ct, m - 1)
   Next
   f = temp
End If
End Function

Sub daus()
Dim n, m As Integer

Open "entrada.dat" For Input As #1
Input #1, n, m
Close #1

Open "sortida.dat" For Output As #1
Print #1, f(n, m)
Close #1

End Sub

Private Sub Form_Load()
daus
End
End Sub
```

7. Resultat d'un partit. Dificultat 2

El partit entre els equips del Barça i del Madrid ha acabat amb el resultat de 3 a 2. Aquest resultat s'ha pogut produït de moltes formes possibles, per exemple, primer marca el Barça, després el Madrid marca dos gols seguits, el Barça empata i, finalment, el Barça marca el definitiu 3 a 2.

Aquesta és una possible forma d'haver arribat a aquest resultat. Heu de pensar de quantes formes diferents s'ha pogut produir aquest resultat, i si arribeu a la conclusió de que hi ha 10 formes possibles ja podeu fer el següent programa:

Imaginem que el partit acaba amb el resultat de **n** a **m**, el programa ha de llegir aquests dos nombres i ha d'escriure totes les formes possibles d'arribar a aquest resultat, escrivint **n** símbols **0** i **m** símbols **1.** Els símbols **0** representen els gols del Barça i els símbols **1** els gols del Madrid.

Exemple:

	fitxer d'entrada 3 2 **fitxer de sortida** 00011 00101 01001 10001 00110 01010 10010 01100 □ cas del exemple de la introducció 10100 11000

8. Sopa de Lletres. Dificultat 2

Les sopes de lletres són un dels passatemps més coneguts. Consisteixen en un quadrat dividit en cel·les. Cada cel·la conté una lletra i amb totes les lletres s'han de formar paraules. Les paraules es poden llegir de dreta a esquerra, d'esquerra a dreta, de dalt a baix, de baix a dalt i en diagonal (tant de dalt a baix com de baix a dalt). Aquest exercici consisteix en buscar una paraula determinada en una taula de lletres entrada a través d'un fitxer. El fitxer d'entrada contindrà **n+1** línies, les **n** primeres seran **n** paraules de **n** lletres i representarà la sopa de lletres. La darrera línia serà una paraula de **m** lletres (**m<n**). El programa haurà de determinar si aquesta última paraula es pot llegir a la sopa de lletres. En cas negatiu, el fitxer de sortida ha de contenir un **0**, en cas positiu, el fitxer de sortida contindrà dos nombres i una lletra. El nombres representaran la fila i la lletra d'aquesta fila (columna) on es troba la primera lletra de la paraula. La lletra següent representarà la direcció on s'ha de llegir la paraula seguint el següent conveni:

- Si la paraula es llegeix d'esquerra a dreta: **D.**
- Si la paraula es llegeix de dreta a esquerra: **E**.
- Si la paraula es llegeix de dalt a baix: **B**.
- Si la paraula es llegeix de baix a dalt: **A.**
- Si la paraula es llegeix en diagonal de dalt a baix i d'esquerra a dreta: **I**
- Si la paraula es llegeix en diagonal de dalt a baix i de dreta a esquerra: **P**
- Si la paraula es llegeix en diagonal de baix a dalt i d'esquerra a dreta: **X**
- Si la paraula es llegeix en diagonal de baix a dalt i de dreta a esquerra: **Z**

Exemple:

C	A	S	T		**fitxer d'entrada:** CAST ZAGS CDSC SEAA CASA **fitxer de sortida:** 11I
Z	**A**	G	S		
C	D	**S**	C		
S	E	A	**A**		

Observació: En el cas de que la paraula es pugui llegir de més d'una forma, el fitxer de sortida contindrà només una de les possibles formes.

9. L'algoritme 3n+1. Dificultat 2

El problema que es planteja aquí és estudiar un dels algoritmes més clàssics no resolts de la ciència de l'Algorítmica: l'algoritme 3n+1. Considerem el següent algoritme:

- *entrar n*
- *imprimir n*
- *si n =1 aleshores ACABA EL PROGRAMA*
- *si n es senar aleshores n:=3n+1*
- *en cas contrari, és a dir, si n es parell, aleshores n:=n/2*
- *tornar a la línia 2*

Per exemple, donat el número *22* el programa imprimiria la següent seqüència de números: *22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1*. Aquesta seqüència rep el nom de **cicle** del número 22. Aquest és un cicle de longitud 16.

És una conjectura no demostrada que aquest algoritme acaba sempre, és a dir, que el cicle de tot nombre enter és un cicle finit.

El problema que es planteja és determinar el cicle de longitud màxima i el de longitud mínima de entre els nombres compresos entre dos llegits en un fitxer.
El fitxer d'entrada consistirà en dos nombres enters positius més petits que 10.000 **(i** i **j**).
El fitxer de sortida haurà de contenir el nombre comprés entre **i** i **j** (ambdós inclosos) que té un cicle més petit, la longitud d'aquest cicle, el nombre comprés entre **i** i **j** (ambdós inclosos) que té un cicle més gran i la longitud d'aquest últim cicle:

Exemple:

5	**6**	**7**	**8**	**9**	**10**		
16	3	22	4	28	5		**fitxer d'entrada:** 5 10 **fitxer de sortida:** 8 4 9 20
8	10	11	2	14	16		

4	5	34	1	7	8		
2	16	17		22	4		
1	8	52		11	2		
	4	26		34	1		
	2	13		17	4		
	1	40		52	2		
		20		26	1		
		10		13			
		5		40			
		16		20			
		8		10			
		4		5			
		2		16			
		1		8			
				4			
				2			
				1			

10. Justificació d'un text. Dificultat 2

La justificació és una de les funcions usuals dels processadors de textos. Consisteix en afegir espais en blanc entre les paraules d'una línia de tal forma que totes les línies del text, incloent-hi els espais en blanc, tinguin el mateix nombre de caràcters.

Ens proposem un programa que, donat **n** línies de text de **m** caràcters o menys cadascuna, afegeixi els espais en blanc entre paraules necessaris per justificar tot el text.

Els espais en blanc s'han de repartir equitativament entre totes les separacions de paraules començant d'esquerra a dreta. Per exemple, si s'ha d'afegir 10 espais en blanc en la següent línia:

aa(1)aaa(1)aaaa(1)aaaa(1)aaaaa(1)a(1)aa

s'afegiran de la següent forma:

aa(3)aaa(3)aaaa(3)aaaa(3)aaaaa(2)a(2)aa

(els nombres entre parèntesi indiquen el nombre d'espais totals entre paraules)

El fitxer d'entrada contindrà dos nombres: el nombre de línies (**n**) i el nombre de caràcters desitjats per cada línia (**m**), a més de les **n** línies, totes elles amb un nombre de caràcters més petit o igual a **m**.

Exemple:

		fitxer d'entrada 3 20 aa aaa aaa aaaa a aa a a aaa aa aaaaa aa aaaa **fitxer de sortida** aa aaa aaa aaaa a aa a a aaa aa aaaaa aa aaaa
aa(1)aaa(1)aaa(1)aaaa(1)a	total 17 caràcters	
aa(1)a(1)a(1)aaa(1)aa	total 13 caràcters	
aaaaa(1)aa(1)aaaa	total 13 caràcters	
aa(2)aaa(2)aaa(2)aaaa(1)a	total 20 caràcters	
aa(3)a(3)a(3)aaa(2)aa	total 20 caràcters	
aaaaa(5)aa(4)aaaa	total 20 caràcters	

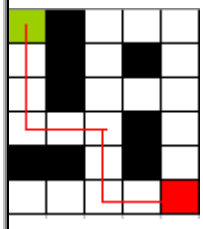
11. El Laberint. Dificultat 3

Imaginem un quadrat dividit en **nxm** quadradets als quals ens referirem per la seva fila i la seva columna. Dels **nxm** quadradets n'hi han **k** que estan marcats i no es poden travessar. El programa ha de trobar el camí **més curt** per passar de la cantonada (**1,1)** a la cantonada (**n,m)** sense passar pels quadradets marcats.

El fitxer d'entrada contindrà els valors de **n** i de **m** així com les coordenades dels **k** quadradets marcats. En el cas de que el camí no existeixi el fitxer de sortida contindrà un 0.

En el cas de que el camí mínim no sigui únic, el fitxer de sortida contindrà un d'ells.

Exemple:

	**fitxer d'entrada** 5 6 1 2 2 2 2 4 3 2 4 4 5 1 5 2 5 4 **fitxer de sortida** 1 1 2 1 3 1 4 1 4 2 4 3 5 3 6 3

	6 4 6 5

12. Màxim rectangle. Dificultat 2

Donada una taula rectangular de nombres enters positius i negatius, es demana trobar un subrectangle que contingui la màxima suma possible dels nombres continguts a les cel·les. El subrectangle demanat pot ser tan petit com una sola cel·la o tan gran com la taula sencera.
El fitxer d'entrada contindrà dos nombres **n** i **m** que seran les dimensions de la taula i **n** files amb **m** nombres enters cadascuna amb els continguts de cada cel·la. El fitxer de sortida contindrà les coordenades superior esquerra e inferior dreta del subrectangle demanat així com la seva suma.

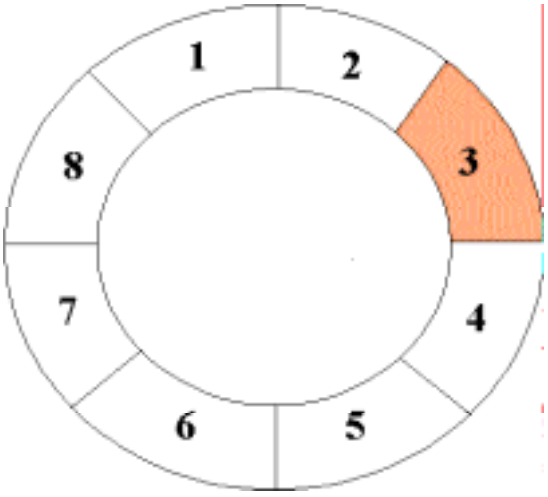
0 −2 −7 0 9 2 −6 2 −4 1 −4 1 −1 8 0 −2 el subrectangle buscat és 9 2 −4 1 −1 8	**fitxer d'entrada:** 4 4 0 -2 -7 0 9 2 -6 2 -4 1 -4 1 -1 8 0 –2 **fitxer de sortida:** 2 1 4 2 15

Annex VIII. 1ª Pre-Olimpíada Informàtica Catalana. Exercicis de la fase prèvia

Exercici 1: SALTS

ENUNCIAT

Imaginem un camí rodó dividit en ***n*** caselles numerades de l'*1* al ***n*** començant per una casella determinada i seguint el sentit de les agulles del rellotge.



Si comencem des de la casella m_1 i fem salts de la mateixa magnitud $k > 0$ en el sentit de les agulles del rellotge, tindrem una seqüència periòdica de caselles de període més petit o igual que ***n.*** Per exemple, si $n = 8$, $m_1 = 3$ i $k = 5$, la seqüència, que tindrà període *8*, serà:

3, 8, 5, 2, 7, 4, 1, 6, 3, 8, 5, 2, 7, 4, 1, 6, 3, 8, 5, 2, 7, 4, 1, 6, ...

És fàcil comprovar que, si ***n*** i ***k*** són primers entre ells, la seqüència tindrà període ***n***, en cas contrari, si ***n*** i ***k*** tenen divisors comuns diferents d'*1*, la seqüència tindrà període més petit que ***n*** (concretament un divisor de ***n***). Per exemple, si $n = 8$, $m_1 = 3$ i $k = 6$, la seqüència, que tindrà període *4*, serà:

3, 1, 7, 5, 3, 1, 7, 5, 3, 1, 7, 5 , ...

Ens interessa conèixer el nombre mínim de salts necessaris per arribar des de la casella m_1 a la casella m_2 amb salts de mida ***k.*** S'ha de tenir en compte que si ***n*** i ***k*** són primers entre ells, el problema té sempre solució, - aquest nombre de salts serà igual o inferior a ***n -*** i en el cas de que ***n*** *i* ***k*** tinguin divisors comuns podria ser que mai s'arribés a la casella m_2.

Es demana: *Fer un programa que llegeixi les dades* n, m_1, m_2 *i* ***k*** *i calculi el nombre mínim de salts necessaris per arribar des de la casella* m_1 *a la casella* m_2, *fent salts de mida* ***k*** *en el sentit de les agulles del rellotge, en un camí rodó dividit en* ***n*** *caselles.*
El programa ha de preveure el cas en que no es pugui assolir la casella m_2. *També ha de considerar el cas* $m_1 = m_2$, *cas en el qual es considerarà que el nombre mínim de salts serà 0.*

Les dades d'entrada es llegiran d'un fitxer de text anomenat:

SALTS.IN

Aquest fitxer contindrà els quatre números enters estrictament positius :***n***, m_1, m_2 i ***k***, un en cada línia i en aquest ordre. Tots els números seran més petits o igual a 40.000.
Els nombres m_1, i m_2 seran més petits o iguals a ***n***. No fa falta la comprovació d'aquesta circumstància.

Les dades de sortida s'escriuran en el fitxer de text:

SALTS.OUT

En aquest fitxer, el programa escriurà un nombre enter més petit o igual a ***n,*** en el cas de que es pugui assolir la casella m_2, o bé la paraula ***impossible*** en el cas de que no es pugui.

EXEMPLES:

SALTS.IN	SALTS.OUT
8 3 7 5	4
8 3 3 5	0
8 3 8 6	

INFORME DE RESOLUCIÓ

A continuació es presenta una possible solució al problema. La part important del codi està marcat amb fons gris:

ct és una variable de tipus ***int*** que fa de comptador de passos. El bucle **for** s'executa fins que **m2==(m1+k*ct)%n.** Quan es troba aquesta coincidència, s'escriu al fitxer de sortida el nombre de passos (**ct**) i surt amb **exit(0).**

El que els nombres siguin més petits o igual a 40.000 asseguren que fent servir enters de 4 bytes l'operació: **m1+k*ct** sempre serà entera.

Si després de **n** passades pel bucle no s'ha trobat la coincidència, surt del bucle i escriu "impossible" al fitxer de sortida.

```
/* salts.c*/

#include <stdio.h>                    /*FILE, fopen, printf, fprint,...*/
#include <stdlib.h>                   /*exit*/

int main() {
```

```
    FILE *fint, *fout;
    int n,m1,m2,k;                  /* dades llegides*/
              int ct;                               /* comptador*/

              /* s'obren els fitxers d'entrada i de sortida*/
    if((fint=fopen("salts.in","r"))==NULL){
          printf("no es pot obrir el fitxer d'entrada\n");
          exit(1);
    }
     if((fout=fopen("salts.out","w"))==NULL){
          printf("no es pot obrir el fitxer de sortida\n");
          exit(1);
    }

              /* es llegeix les dades del fitxer d'entrada*/
    fscanf(fint,"%d%d%d%d",&n,&m1,&m2,&k);

              for(ct=0;ct<n;ct++){
                        if(m2%n==(m1+k*ct)%n) {
                                  fprintf(fout,"%d\n",ct);
                fclose(fint);fclose(fout);
                                  exit (0);
                        }
              }
              fprintf(fout,"impossible\n");
              fclose(fint);fclose(fout);
}
```

JOCS DE PROVA

0

El primer exemple de l'enunciat

SALTS0.IN	SALTS0.OUT
8 3 7 5	4

1

El segon exemple de l'enunciat

SALTS1.IN	SALTS1.OUT
8 3 3 5	0

2

El tercer exemple de l'enunciat

SALTS2.IN	SALTS2.OUT
8 3 8 6	impossible

3

El cas més petit

SALTS3.IN	SALTS3.OUT
1 1 1 1	0

4

Un cas en el qual k > n.

SALTS4.IN	SALTS4.OUT
8 3 4 12	impossible

5

Un altre cas en el qual k > n.

SALTS5.IN	SALTS5.OUT
8 3 5 9	2

6

Un cas en el qual k = n.

SALTS6.IN	SALTS6.OUT
8 3 4 8	impossible

7

Un altre cas en el qual k = n.

SALTS7.IN	SALTS7.OUT
8 3 3	0

8	

8

Un amb n gran

SALTS8.IN	**SALTS8.OUT**
1652 3 512 125	1101

9

Un altre amb n i k grans

SALTS9.IN	**SALTS9.OUT**
40.000 1111 2222 3333	26667

Exercici 2: PATRONS RECTANGULARS

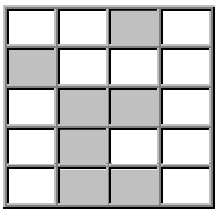
ENUNCIAT

Imagina un rectangle de ***m*** files i ***n*** columnes dividit en quadrats tal com mostra la següent figura.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20

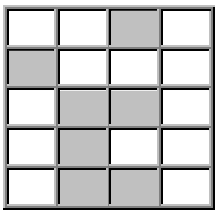
Cada quadrat està enumerat seqüencialment, de forma que el quadrat superior esquerra és el 1, el de la seva dreta el 2 i així successivament fins arribar al inferior dreta que és el ***m*n**.

La següent figura mostra un rectangle amb alguns quadrats marcats:



Ara, considera la regió ***p*q*** que comença a la cantonada superior esquerra de la segona figura. Per exemple, si ***p*** = 2 i ***q*** = 1 llavors obtenim la següent regió:

Aquest patró apareix en tot el rectangle original en quatre posicions més:



Es demana *fer un programa que llegeixi les dades d'un rectangle i d'una regió i calculi quantes vegades apareix la regió en el rectangle original.*

No s'ha de considerar rotacions de la regió. La regió sempre es selecciona des de la cantonada superior esquerra.

Les dades d'entrada estan en el fitxer:

PATREC.IN

La primera línia d'aquest fitxer conté quatre números enters que són els valors per ***m*** (1≤ ***m***≤ 20), ***n*** (1≤ ***n***≤ 20), ***p*** (1≤ ***p***≤ ***m***) i ***q*** (1≤ ***q***≤ ***n***) respectivament. La següent línia és x, el nombre de quadrats marcats (0≤ ***x***≤ ***m*n***) . Cada una de les següents ***x*** línies indica la posició d'un quadrat marcat, utilitzant la numeració mostrada a la primera figura.

Les dades de sortida s'escriuran al fitxer de text:

PATREC.OUT

en el qual hi ha una sola línia que conté un número enter, el nombre de vegades en què apareix la regió dins del rectangle, incloent-hi la regió original de la cantonada superior esquerra.
S'ha de tenir en compte que un quadrat pot formar part de més d'una regió.

EXEMPLES:

PATREC.IN	**PATREC.OUT**
5 4 2 1 **7** **3** **5** **10** **11** **14** **18** **19**	**4**
4 5 2 2 **2** **2** **5**	**2**

INFORME DE RESOLUCIÓ

Donat el següent rectangle:

n

m														
	.	.	*	*	.	.	*	.	.	.	.	.	.	*
	*	.	*	*	*	.	.	.	.	.	.	*	*	*
	*	.	*	.	*	*	.	*	*	*	.	*	.	*	.
	*	.	.	*	.	.	.	.	*	.	.	.	.	.	.
	.	.	*	.	.	*	*	*	.	.	*	*	*	*	.
	.	*	.	.	.	*	.	*	*	*	*	.	*	*	.
	.	.	*	*	*	.	*	*	*	.	.	*	*	.	.
	*	.	*	*	.	*	*	*	.	.	*	*	.	*	.
	*	.	*	.	.	*	.	.	.	*	*	*	.	.	.
	.	.	.	*	*	*	.	*	.	*	*	.	*	.	*
	*	.	.	*	*	.	*	*	.	*	.	.	.	.	.

La regió és:

q

p			
	.	.	*
	*	.	*

Per trobar el nombre de vegades en què apareix la regió en el rectangle, situarem el quadrat superior esquerra de la regió sobre de cada quadrat del rectangle, si els quadrats que formen a la regió són iguals als que té a sota vol dir que hem trobat una coincidència i la comptabilitzem.

Cal tenir en compte que la regió sempre ha d'estar a sobre del rectangle...

```
program patrec;
var r:array[0..19,0..19]of boolean;
    m,n,p,q,x,c,i,j:integer;
    te,ts:text;

procedure Llegirdades;
 var i,j,a:integer;
 begin
  assign(te,'patrec2.in'); reset(te);
  read(te,m,n,p,q);
  read(te,x);
  for i:=0 to m-1 do
   for j:=0 to n-1 do
    r[i,j]:=false;
  for i := 1 to x do
   begin
    read(te,a); a:=a-1;
    r[a div n,a mod n]:=true;
   end;
  close(te);
 end;

procedure EscriuResultat;
 begin
  assign(ts,'patrec.out'); rewrite(ts);
  writeln(ts,c);
  close(ts);
 end;

Function encaixa(fila,col:integer):boolean;
 var ok:boolean;
     i,j:integer;
 begin
  ok:=true;
  for i:=0 to p-1 do
   for j:=0 to q-1 do
    ok:=(r[i,j]=r[i+fila,j+col]) and ok;
  encaixa:=ok;
 end;

Begin
 LlegirDades;

 { algoritme }
 c:=0;              {Comptador del nombre d'ocurrències de la regió.}
 for i:=0 to m-p do {Recorrem tota la taula i verifiquem si la }
  begin             {regió hi encaixa. }
   for j:=0 to n-q do
    begin
     if encaixa(i,j) then c:=c+1;
    end;
   end;
 { fi de l'algoritme }

 EscriuResultat;
end.
```

JOCS DE PROVA

0

PATREC0.IN

L'exemple de l'enunciat.

.	.	*	.
*	.	.	.
.	*	*	.
.	*	.	.
.	*	*	.

5 4 2 1
7
3
5
10
11
14
18
19

PATREC0.OUT

4

1

PATREC1.IN

El cas més petit.

*

1 1 1 1
1
1

PATREC1.OUT

1

2

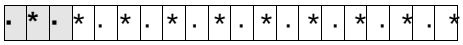
PATREC2.IN

Una Columna.

*
.
*
.
*
.
*
.
*
.

20 1 3 1
10
1
3
5
7
9
11
13
15
17
19

PATREC2.OUT
9

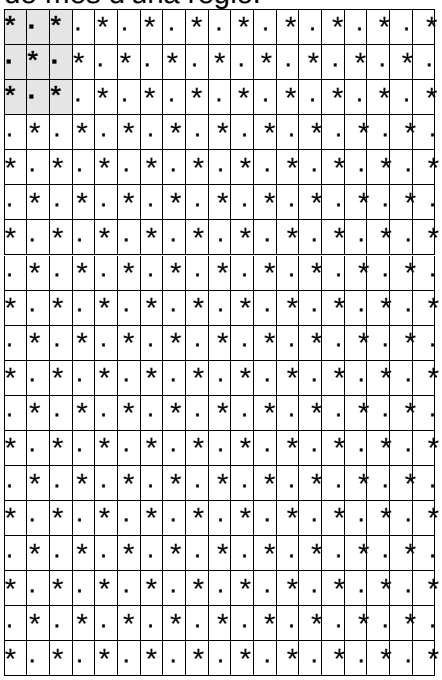
3
PATREC3.IN
Una fila.


1 20 1 3
10
2
4
6
8
10
12
14
16
18
20

PATREC3.OUT
9

4
PATREC4.IN

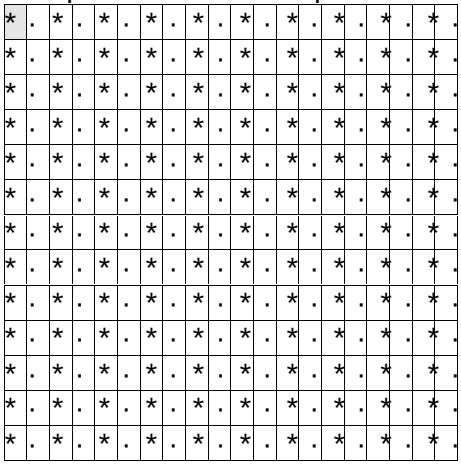
Taula gran amb els quadrats negres en posició senar. Hi han quadrats que formen part de més d'una regió.

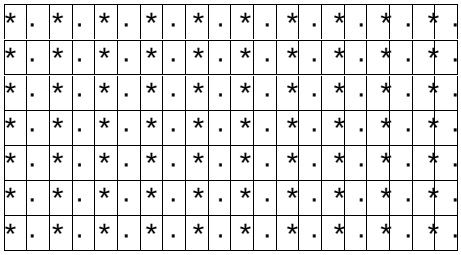


19 19 3 3
181
1
3
5
7
...
359
361

PATREC4.OUT
145

5
PATREC5.IN
Comptar el nombre de quadrats com el de la posició (1,1).

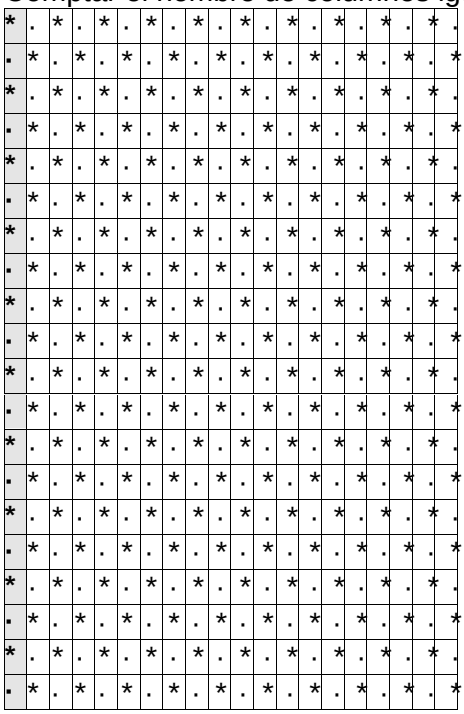




20 20 1 1
200
1
3
5
7
...
397
399

PATREC5.OUT
200

6
PATREC6.IN
Comptar el nombre de columnes iguals a la primera.



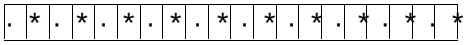
20 20 20 1
200
1
3
5
7

9
11
13
15
17
19
22
24
26
28
30
32
34
36
38
40
41
43
45
...
392
394
396
398
400

PATREC6.OUT
10

7
PATREC7.IN
Comptar el nombre de files iguals a la primera.

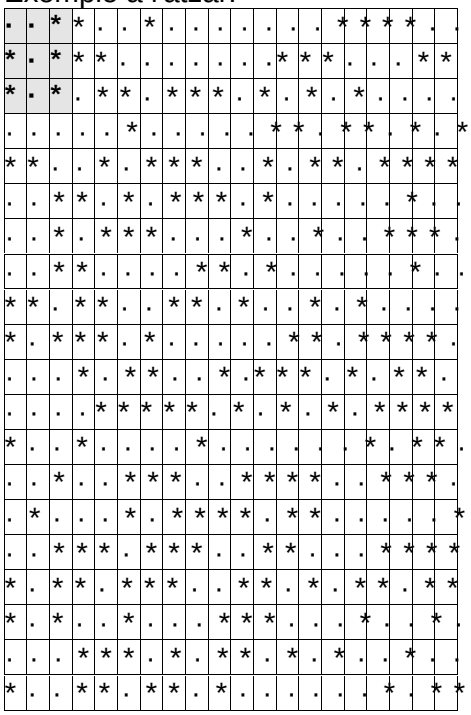
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.
.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*
*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.	*	.



20 20 1 20
200
1
3
5
7
...
396
398
400

PATREC7.OUT
10

8
PATREC8.IN
Exemple a l'atzar.

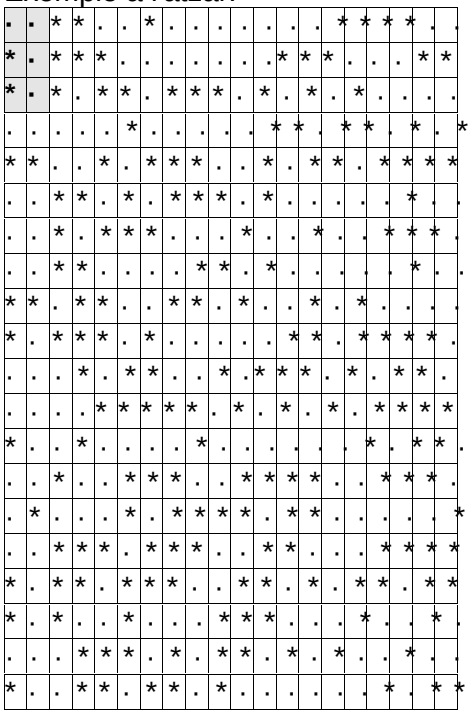


20 20 3 3
188
3
4
7
15
16
17
...
397
399

400

PATREC8.OUT
3

9
PATREC9.IN
Exemple a l'atzar.



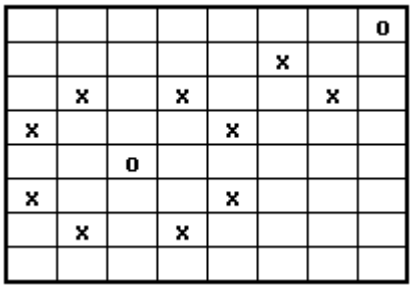
20 20 3 2
188
3
4
7
15
16
17
...
397
399
400

PATREC9.OUT
9

Exercici 3: EL CAVALL DEL ESCACS

ENUNCIAT

El moviment del cavall en el joc dels escacs és un moviment amb propietats molt curioses. Recordem que aquest moviment és sempre una combinació de dues caselles en una direcció seguit d'una casella en una direcció perpendicular, sempre que aquesta casella final estigui dintre del tauler. D'aquesta forma, per un cavall situat prop del centre del tauler hi ha 8 possibles moviments, en canvi, per un cavall situat just en una cantonada n'hi ha només 2, com es pot contemplar a la figura:



El problema que ens plantegem és el següent: *Donades dues caselles, una casella inicial i una casella final, trobar el nombre mínim de moviments que ha de fer un cavall per anar de la primera a la segona.*

Fixarem la següent notació: cada casella es representa per dos nombres enters que representen la fila i la columna. La casella superior esquerra serà la casella (0,0) i la casella inferior dreta serà la casella (7,7).

00	01	02	03	04	05	06	07
10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37
40	41	42	43	44	45	46	47
50	51	52	53	54	55	56	57
60	61	62	63	64	65	66	67
70	71	72	73	74	75	76	77

El **fitxer d'entrada** serà un fitxer de text anomenat:

CAVALL.IN

que contindrà a la primera línia, separades per un espai, la fila i la columna de la casella inicial, i a la segona línia, també separades per un espai, la fila i la columna de la casella final. Les dades de les caselles inicials i finals estan dintre del tauler (no és necessari fer la comprovació).

El **fitxer de sortida** serà un fitxer de text anomenat:

CAVALL.OUT

amb un sol nombre enter més gran o igual a **0** que indicarà el nombre mínim de moviments per arribar de la casella inicial a la casella final. En el cas de que ambdues caselles coincideixin, el nombre mínim de moviments serà **0**.

EXEMPLES:

CAVALL.IN	CAVALL.OUT
0 1 5 4	4
2 3 2 3	0

INFORME DE RESOLUCIÓ

Per il·lustrar aquest problema suposem que la casella inicial fos la casella (0,1). Aquesta casella determina en el tauler una classificació de totes les altres caselles tal i com es pot veure a la següent figura:

3	0	3	2	3	2	3	4
2	3	2	1	2	3	4	3
1	2	1	4	3	2	3	4
2	3	2	3	2	3	4	3
3	2	3	2	3	4	3	4
4	3	4	3	4	3	4	5
3	4	3	4	3	4	5	4
4	5	4	5	4	5	4	5

El número de cada casella indica el mínim nombre de moviments que necessita el cavall per anar-hi des de la casella (0,1). En aquest cas, a totes les caselles es pot arribar en 5 moviments o menys –es pot veure fàcilment que per qualsevol posició inicial sempre es pot assolir qualsevol altra casella en 6 moviments o menys–.

Per fer el programa farem servir una llista de caselles on ficarem:
en primer lloc, la casella inicial,
a continuació les caselles que es poden visitar amb un moviment,
a continuació les caselles que es poden visitar amb dos moviments,
i així successivament:

Aquí tenim un esquema d'aquesta llista:

pos nº	0	1	2	3	4	5	6	7	8	9	10	11
casella	0 1	1 3	2 2	2 0	0 5	2 5	3 4	3 2	2 1	1 0	0 3	...
mov	0	1	1	1	2	2	2	2	2	2	2	...
apunt:		a		b							d	

En aquesta llista tindrem 3 apuntadors que anomenarem a, b i d.

a serà la primera casella que es pot visitar en ***ct-1*** moviments.
b serà la darrera casella que es pot visitar en ***ct-1*** moviments.
d serà la última casella apuntada. Serà una variable global.

Per trobar les caselles que es poden visitar en un mínim de ***ct*** moviments, explorem tots els moviments del cavall possibles de les caselles des de la posició ***a*** a la posició ***b***. Per cada moviment del cavall trobada explorem si no és la casella final buscada, si està dintre del tauler i si no pertany ja a la llista. Si és així, l'apuntem a la llista i augmentem en ***1*** l'apuntador ***d.***

Per exemple, si estem explorant les caselles que es poden visitar en un mínim de 2 moviments tindrem: ***a =*** 1, ***b =*** 3 *i* ***d*** anirà canviant des de 4 fins a 15

Una vegada acabada l'exploració de totes les caselles des d'***a*** fins a ***b,*** fem a = b + 1 i b = d i comencem l'exploració per trobar les caselles que es poden visitar en ***ct +1*** moviments.

En el programa es fan servir quatre funcions auxiliars:

explora_fila(int f, int mov) torna la fila després de fer el moviment ***mov*** (dels 8 possibles del cavall) des d'una posició que té fila **f**.
explora_colu(int c, int mov) torna la columna després de fer el moviment ***mov*** (dels 8 possibles del cavall) des d'una posició que té columna **c**.

fora(int f, int c) torna 1 si la casella (f,c) està fora del tauler, en cas contrari torna un 1.

ocupat (int f, int c) comprova a la llista global de caselles si es troba la casella (f,c).

El codi és el següent:

```
//cavall.c
#include <stdio.h>
#include <stdlib.h>

int explora_fila(int f, int mov);  // nova fila després del moviment mov
int explora_colu(int c, int mov);          // nova columna després del moviment mov
int fora(int f, int c);            // està fora del tauler?
int ocupat (int f, int c);                 // ja hem arribat abans a aquesta casella?

int n[2][64];                                       //llista de caselles ocupades
int d;                                              //apuntador global de la llista

int main(){

    FILE *fint, *fout;
    int o1,o2,f1,f2;            //o1,o2 casella inicial, f1,f2 casella final
    int a,b;                    //apuntadors a la llista de caselles ocupades
    int f, c;                   //fila i columna de la casella a estudiar
    int ct,ct1,ct2;             //diferents comptadors


   // s'obren els fitxers d'entrada i de sortida i es llegeixen les dades

    if((fint=fopen("c:/debug/cavall.in","r"))==NULL){
          printf("no es pot obrir el fitxer d'entrada\n");
          exit(1);
    }

    if((fout=fopen("c:/debug/cavall.out","w"))==NULL){
          printf("no es pot obrir el fitxer de sortida\n");
          exit(1);
    }

    fscanf(fint,"%d %d\n%d %d",&o1,&o2,&f1,&f2);
```

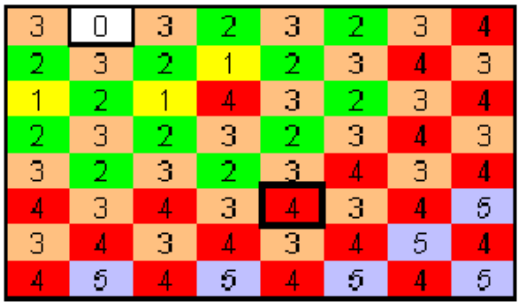
```
   //*******************************************************


   // algoritme principal

    a=0;b=0;d=0;
    n[0][b]=o1;n[1][b]=o2;

              if (o1==f1 && o2==f2) {                    //sí coincideixen la casella inicial
                        fprintf(fout,"%d\n",0);          // i la casella final, escriu un 0 i
                        fclose(fint);fclose(fout);
                        exit(0);                         //surt
              }

              for(ct=1;d<63;ct++){
                         for (ct1=a;ct1<=b;ct1++){
                                   for (ct2=0;ct2<8;ct2++){
                                             f=explora_fila(n[0][ct1],ct2);
                                             c=explora_colu(n[1][ct1],ct2);
                                             if(!fora(f,c)) {
                                                       if(!ocupat(f,c)) {
                                                                 if (f==f1 && c==f2) {
                                                                           fprintf(fout,"%d\n",ct);
                                                                           fclose(fint);fclose(fout);
                                        exit(0);
                                                                 }
                                                                 d=d+1;
                                                                 n[0][d]=f;n[1][d]=c;
                                                       }
                                             }
                                   }
                         }
                         a=b+1;b=d;

              }
}
//*************************************************************

int explora_fila(int f, int mov){
              switch (mov){
              case 1:
              case 2:
                        return f-2;break;
              case 0:
              case 3:
                        return f-1;break;
              case 7:
              case 4:
                        return f+1;break;
              case 5:
              case 6:
                        return f+2;break;                                                                                    }
}
//*************************************************************

int explora_colu(int c, int mov){
              switch (mov){
              case 7:
              case 0:
                        return c-2;break;
              case 1:
              case 6:
                        return c-1;break;
              case 2:
              case 5:
                        return c+1;break;
              case 3:
              case 4:
                        return c+2;break;                                                                                    }

}
```

```
//************************************************************

int fora(int f, int c){
        if (f<0 ||f>7 || c<0 ||c>7) return 1;
        else return 0;
}

//************************************************************

int ocupat (int f, int c){
        int ct3;
        for (ct3=0;ct3<=d;ct3++)        {
                if (f==n[0][ct3] && c==n[1][ct3]){
                        return 1;
                break;
        }
}
        return 0;
}

//************************************************************
```

JOCS DE PROVA

0

El primer exemple de l'enunciat



CAVALL0.IN	CAVALL0.OUT
0 1 5 4	4

1

El segon exemple de l'enunciat

La casella inicial és igual a la casella final

CAVALL1.IN	CAVALL1.OUT
2 3 2 3	0

2

Dos caselles separades per un salt de cavall només:

La casella inicial és igual a la casella final

CAVALL1.IN	CAVALL1.OUT

2 3 4 4	1

3

Dos caselles properes però separades amb 4 moviments

CAVALL3.IN	CAVALL3.OUT
0 0 1 1	4

4

Dos caselles contigües

CAVALL4.IN	CAVALL4.OUT
2 3 2 4	3

5

El cas més allunyat

CAVALL5.IN	CAVALL5.OUT
0 0 7 7	6

6

El mateix que l'anterior canviant les posicions inicials i finals

CAVALL6.IN	CAVALL6.OUT
7 7 0 0	6

7

El cas de 5 posicions.

CAVALL7.IN	CAVALL7.OUT
0 1 7 1	5

8

La posició final és sota la posició inicial

CAVALL8.IN	CAVALL8.OUT
2 5 4 1	2

9

La posició inicial és sota la posició final

CAVALL9.IN	CAVALL9.OUT
6 2 0 0	4

Annex IX. 1ª Pre-Olimpíada Informàtica Catalana. Exercicis de la fase final

Exercici 1: LOTS DE LLIBRES

En una llibreria s'han rebut ***n*** lots de llibres en ***n*** dies diferents. El primer lot contenia m_1 llibres, el segon lot m_2 llibres, i així successivament fins al lot ***n*** que contenia m_n llibres. Quan arriben llibres a la llibreria són registrats immediatament. S'assigna un número enter diferent a cada llibre. Els llibres del primer lot tenen assignats els números: $1,2,\ldots m_1$. Els llibres del segon lot tenen assignats els números: $m_1+1, m_1+2, \ldots, m_1+m_2$ i així fins als llibres del lot ***n*** que tenen assignats els números: $m_1+m_2+\ldots+m_{n-1}+1, m_1+m_2+\ldots+m_{n-1}+2, \ldots, m_1+m_2+\ldots+m_{n-1}+m_n$.

Pot ser la notació us pugui despistar una mica, però podeu pensar que s'han rebut un total de $m_1+m_2+\ldots+m_{n-1}+m_n$ llibres i s'han registrat tots, de forma ordenada, amb un número de l'***1*** al número total de llibres.

Es demana *fer un programa que, donat un número de registre, ens indiqui en quin lot ha vingut.*

El **fitxer d'entrada** serà un fitxer de text anomenat **LOTS.IN,** que consistirà en ***n+2*** números enters, un a cada línia. El primer serà el nombre ***n*** de lots, els ***n*** números restants seran el nombre de llibres que té cada lot, és a dir, els números $m_1, m_2,\ldots,m_n$. Per últim, hi haurà un número enter comprès entre ***1*** i $m_1+m_2+\ldots+m_n$.

El **fitxer de sortida** s'anomenarà **LOTS.OUT** i haurà de contenir un únic número enter entre ***1*** i ***n*** que serà el número de lot en què es troba el llibre.

EXEMPLE:

LOTS.IN	LOTS.OUT
3 10 12 4 17	2

Exercici 2: SUMES DE VEÏNS

Donada una taula ***n x n,*** com la de la figura adjunta, amb nombres enters, es tracta de construir una altra taula amb el contingut de totes les sumes dels seus veïns.

1	2	5	4	5
2	4	8	6	4
2	5	6	5	4
2	5	4	5	2
1	4	0	5	8

Anomenarem veïns a les caselles que estiguin a sobre, a sota, a la dreta, a l'esquerra o en diagonal. S'ha de tenir en compte que no totes les cel·les tenen el mateix nombre de veïns. Per exemple, la casella de dalt a l'esquerra té només tres veïns mentre que una casella central en té vuit, com es pot veure a la següent figura:

*	2			
2	4			
		6	5	4
		4	*	2
		0	5	8

En el exemple proposat, el resultat de la primera casella hauria de ser: 2 + 4 + 2 = 8
El resultat de la casella de la quarta fila, quarta columna, hauria de ser: 6 + 5 + 4 + 4 + 2 + 0 + 5 + 8 = 34

El **fitxer d'entrada** serà un fitxer de text anomenat **SUMES.IN** que contindrà a la primera línia el nombre ***n*** de files i de columnes de la graella (***n*** $\leq$ 20). A partir de la segona línia hi estaran els nombres enters de les caselles. Aquests nombres, que podran ser positius o negatius, estaran ordenats de forma que cada fila de nombres s'escriguin en línies diferents i cada línia contingui els ***n*** nombres de la fila corresponent separats per un espai.

El **fitxer de sortida** serà un fitxer de text anomenat **SUMES.OUT,** que haurà de tenir una estructura semblant a les ***n*** últimes línies del fitxer d'entrada. A cada línia s'escriuran els ***n*** nombres de la fila corresponent separats per un espai.

EXEMPLE:

SUMES.IN	**SUMES.OUT**
5 1 2 5 4 5 2 4 8 6 4 2 5 6 5 4 2 5 4 5 2 1 4 0 5 8	8 20 24 28 14 14 31 37 41 24 18 33 42 39 22 17 24 35 34 27 11 12 23 19 12

Exercici 3: LA TROBADA DELS AMICS

n amics s'han trobat un dia en un bar d'un petit poble dels Pirineus catalans. Feia molt de temps des de l'última vegada. El primer dels ***n*** amics diu als altres que està tan content de la trobada que tornarà al bar cada m_1 dies. El segon dels amics diu que tornarà al bar cada m_2 dies, i així successivament fins a l'últim dels amics, que tornarà al bar cada m_n dies. Per exemple, si s'han trobat el dia 8 de gener i un diu que hi anirà cada 3 dies vol dir que hi tornarà l'11 de gener, el 14 de gener, el 17 de gener, etc.

El problema que ens plantegem és *trobar quants dies han de passar per tal que es tornin a trobar tots els amics.*

El **fitxer d'entrada,** que serà un fitxer de text anomenat **AMICS.IN,** contindrà el nombre ***n*** d'amics i els ***n*** nombres $m_1, m_2, ...m_n$, cada nombre en una línia diferent. Tots els nombres del fitxer d'entrada seran nombres enters positius.

El **fitxer de sortida** haurà de ser un fitxer de text anomenat **AMICS.OUT** i contindrà un número enter corresponent als dies que han de passar per tal que tots els amics tornin a coincidir en el bar.

EXEMPLE:

AMICS.IN	**AMICS.OUT**
5 10 12 20 25 50	300

Exercici 4: MISSATGES SECRETS

Busquem un programa que permeti encriptar i desencriptar missatges. El mètode que farem servir serà el següent:

Cada paraula de ***n*** lletres es substituirà per una altra paraula de les mateixes lletres agafant-les en el següent ordre: primera, última, segona, penúltima, tercera, antepenúltima, etc.

Per exemple, el següent missatge: AQUEST ÉS UN MISSATGE SECRET
serà substituït per: ATQSUE ÉS UN MEIGSTSA STEECR

Amb aquest mètode, les paraules d'una i dos lletres no canvien.
El **fitxer d'entrada** serà un fitxer de text anomenat **MISS.IN** que contindrà, en la primera línia, una cadena de caràcters formada per paraules separades per espais. Aquesta cadena no serà superior a 255 caràcters.

A la segona línia hi haurà un ***0*** o un ***1***. Un ***0*** representa que s'ha d'encriptar i un ***1*** representa que s'ha de desencriptar.

El **fitxer de sortida** serà un fitxer de text anomenat **MISS.OUT** que contindrà una sola línia amb el missatge llegit encriptat o desencriptat, segons indiqui el número de la segona línia del fitxer d'entrada.

EXEMPLE1:

MISS.IN	**MISS.OUT**
AQUEST ÉS UN MISSATGE SECRET 0	ATQSUE ÉS UN MEIGSTSA STEECR

EXEMPLE2:

MISS.IN	**MISS.OUT**

ATQSUE ÉS UN MEIGSTSA STEECR 1	AQUEST ÉS UN MISSATGE SECRET

Exercici 5: UNA CARRETERA AMB REVOLTS

Tothom prefereix les carreteres amb pocs revolts. Un enginyer ha de dissenyar una carretera que uneixi dues ciutats amb el menor nombre de revolts possible sense importar la longitud de la carretera. Per tal de facilitar el disseny, l'enginyer va dibuixar un mapa amb els accidents naturals per on no podia passar la carretera, com poden ser muntanyes i barrancs. Va quadricular el mapa i va marcar aquests accidents naturals de color negre. La carretera només pot anar d'un quadre a un altre del mapa si tenen en comú un costat o una cantonada, i és clar, no pot passar per cap accident natural (quadre negre).

Direm que hi ha una corba a la carretera en el moment que acaba un tram recte i en comença un altre.

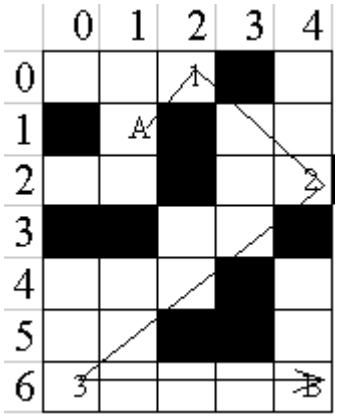
Cada quadre del mapa s'identifica per les seves coordenades, primer la fila i, després la columna. Les files es numeren de dalt a baix començant per la fila 0, i les columnes d'esquerra a dreta, començant també per la columna 0.

Problema
Escriu un programa que donat un mapa amb els accidents naturals, trobi el menor nombre de revolts possible que tingui una carretera que s'inicii en el quadre on es troba la ciutat A i acabi en el quadre on es troba la ciutat B.
El **Fitxer d'entrada** serà un fitxer de text anomenat **REVOLTS.IN.** A la primera fila hi trobareu dos enters **N** i **M** que són el nombre de files i columnes del mapa
(1 □ N,M □ 50). En cadascuna de les **N** files restants hi trobareu **M** números, que poden ser **1** o **0**. L'**1** vol dir que en aquest quadre hi ha un accident natural i, per tant, no pot passar la carretera, i un **0** vol dir que sí que hi pot passar. A la següent línia del fitxer (la penúltima) trobareu les coordenades (fila, columna) de la ciutat A i a la última línia, les coordenades de la ciutat B

El **fitxer de sortida** serà un fitxer de text anomenat **REVOLTS.OUT** que haurà de contenir un nombre enter corresponent al menor número de corbes que pot tenir una carretera entre les ciutats A i B.

EXEMPLE

Al següent mapa de 7 files i 5 columnes, la ciutat A es troba al quadre (1,1) i la ciutat B al quadre (6,4). S'han posat diversos accidents naturals (els quadres negres). El menor número de trams rectes possible per anar d'A a B és 4, per tant, el menor número de corbes és 3. Una possible carretera amb 3 revolts està dibuixada sobre el mapa.



REVOLTS.IN	REVOLTS.OUT
7 5 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 1 6 4	3

NOTA: en tots los casos de prova es pot dissenyar almenys una carretera