Seqüències i Esquemes

Objectius:
- **Detectar si un problema es correspon amb un recorregut o una cerca.**
- **Implementar programes que facin un recorregut.**
- **Implementar programes que facin una cerca.**
- **Llegir seqüències de més d'un element.**
- **Determinar quina és la condició correcta per a l'últim element.**

(Objectius: 3.1.2 i 3.2)

1. Els esquemes

Com a recordatori, en la taula següent es mostren els 2 esquemes típics d'algorismes que treballen amb seqüències:

Recorregut

```
<instruccions_inicials>
<obtenir_1r_element>
while ( <no últim_element> ) {
  <tractar_element>
  <obtenir_seguent_element>
}
<instruccions_finals>
```

Cerca

```
bool trobat = false;
<instruccions_inicials>
<obtenir_1r_element>
while ( <no últim_element> && ! trobat ) {
  if ( <propietat_element> ) trobat = true;
  else <obtenir_seguent_element>;
}
if ( trobat ) <instruccions_trobat>;
else <instruccions_NO_trobat>;
```

Els esquemes són unes "plantilles" a on cal omplir els buits, marcats en blau. Tots dos esquemes suposen l'existència d'un seguit d'elements i el que diferencia els dos esquemes és la necessitat de recórrer tots els elements o no. Aquest és un punt molt important. En el cas del recorregut, es miren tots els elements. Per calcular el màxim d'una seqüència, per exemple, no es pot deixar de mirar cap element (el màxim podria estar just al final). En el cas de la cerca, si l'element de la seqüència que es busca es troba a meitat del càlcul, el bucle es pot abandonar i donar directament el resultat. En els problemes que tenen a veure amb una cerca, el resultat sol ser un Booleà, també.

Els tipus de seqüències amb que treballem sempre solen tenir un element final, que indica la finalització de la seqüència, que es diu sentinella. Si no, típicament disposem d'alguna condició per saber que la seqüència s'ha acabat.

2. Cerca un enter en una seqüència

Dissenyeu un algorisme que faci una cerca d'un enter (introduït per teclat) en una seqüència d'enters entrats per teclat i acabada amb -1 .

3. Fibonacci

Dissenyeu un algoritme que calculi la sèrie de Fibonacci fins un valor entrat per teclat (el valor ens dirà el número d'ordre de l'últim número de la sèrie que s'ha de visualitzar).

4. La identitat del resultat

Examina el següent programa:

```
#include <iostream>
using namespace std;

int main()
{
  float p, ac;
  cin >> p;
  ac = p;
  while ( p != 0.0 ) {
    if ( p > ac ) ac = p;
    cin >> p;
  }
  if ( ac != 0.0 )
    cout << ac;
}
```

Aquest programa t'ha de ser familiar (si no és així, mira d'entendre què fa i inclús estudiar-lo fins a aprendre-te'l). Identifica a quin esquema correspon, primer. Podries indicar qui és el sentinella, en aquest programa?

Ara descarrega el fitxer **`bcn_temp.txt`** d'Atenea, que és bàsicament una llista de temperatures i poblacions. Fes un programa, llavors, que faci el mateix càlcul que fa el programa de dalt (amb les temperatures), però el programa no doni com a resultat una temperatura sinó *quina* població tenia aquesta temperatura. Fixa't que per llegir els elements de la seqüència, hauràs de fer un **`cin`** doble, com aquest:

```
cin >> temp >> ciutat;
```

Determina doncs, quina és la ciutat.

5. Recordar l'anterior

Ara farem un programa que, donada una seqüència de números (acabada en 0), diu per a cada un si la seqüència és creixent, decreixent o constant. Per exemple:

```
5
constant
6
creixent
7
creixent
7
constant
2
decreixent
0
```

Com sempre, la part taronja és el que escriu l'usuari, i la resta el que treu per pantalla el programa. La seqüència és "`5 6 7 7 2 0`" i si t'hi fixes, el primer número no en té cap abans, i en canvi el programa diu "constant". O sigui que considerem (ens ho "inventem") que el primer cop, en llegir un número, la seqüència és constant.

Primer, creem un esquema de recorregut que tingui el mínim d'elements, però que funcioni:

```
#include <iostream>
using namespace std;

int main()
{
  int num;
  cin >> num;
  while ( num != 0 ) {

    cin >> num;
  }
}
```

Aquest programa no fa res de bo, però ja té l'esquema correcte. Això és molt important, ja que només ens resta "omplir" el que falta sense tocar això. Aquest programa, ara per ara, tracta amb una seqüència entrada per l'usuari, que és d'enters i que acaba en un zero. El més important és que el programa funciona, no es queda encallat en un bucle infinit, etc.

Segon, fes el codi necessari que compari en nombre **num** amb un altre que direm **num_ant** i escrigui per pantalla "creixent", "decreixent" o "constant". A la variable num_ant suposem que hi ha en nombre anterior en la seqüència. Ara canvieu el següent codi afegint la comparació a la zona taronja:

```
#include <iostream>
using namespace std;

int main()
{
  int num;
  cin >> num;

  while ( num != 0 ) {
```

```
    cin >> num;
  }
}
```

El programa actual no compila, ja que la variable **num_ant** no existeix, però això és el que farem com a última cosa.

Tercer, arregla el programa de tal manera que declari una variable **num_ant**, i omple les dues instruccions en blau per tal que el programa funcioni. La qüestió és que:
1) En la primera iteració, el programa mostri "constant".
2) En les altres iteracions, **num_ant** ha de contenir el valor que tenia num en la iteració anterior.

6. Repetits

Fent servir el que has après en l'apartat anterior, fes un programa que compti quantes vegades succeeix que una seqüència d'enters acabada en 0 tingui elements consecutius repetits (per exemple, amb la seqüència "`1 3 14 5 5 2 6 -4 -5 9 9 8 7 7 6 9 15 3 -1 0`", el programa ha de dir 3).

Fixa't com canviaria el programa si en comptes de demanar *quants* repetits hi ha, et demanés només *si hi ha repetits o no*. Quin esquema tenen cada una de les versions?

7. Suma < 100

Aquesta secció tracta d'un programa que fa el següent: determina si la suma d'una seqüència de números reals *positius* és menor que 100. Clarament, la resposta és "si" o "no". Per altra banda, com que els números sempre són positius, la suma anirà creixent. Això és important, ja que si en un moment donat del programa, veiem que la suma ha passat de 100, ja sabem que mai podrà tornar a baixar, i podem dir "no" com a resultat amb tota seguretat.

Comencem amb l'esquema de cerca:

```
#include <iostream>
using namespace std;

int main()
{
  bool trobat = false;
  float x;
  cin >> x;
  while ( x > 0.0 && ! trobat ) {
    if ( x == 5.0 ) trobat = true;
    else {
      cin >> x;
    }
  }
  if ( trobat ) cout << "Si" << endl;
  else cout << "No" << endl;
}
```

Aquest programa no fa el que es demanava (què fa, exactament?), tal com ha passat abans, però ja té l'esquema correcte. Això és molt important, ja que evitem problemes (com bucles infinits), i ens ajuda a focalitzar en el que queda per fer (canviar la condició de l'if que està en blau per una de correcta, i tot allò que es necessiti per a la condició). El programa processa una seqüència de reals positius correctament, ara per ara, i dóna un resultat de "si" o "no". Afegeix, doncs, el que calgui per acabar-lo.

8. Seqüència creixent?

Ara fes un programa, inspirant-te en els que has fet fins ara (intenta escriure'l sencer i no fent "cortar y pegar"!!), que determini si una seqüència de reals positius és creixent. Si el programa rep "`0.1 0.2 0.3 0.4 0.5 -1.0`" ha a dir "si". Si el programa rep "`0.1 0.5 1.0 2.0 0.2 0.4 0.6 -1.0`" ha de dir "no". Pensa si pots donar una resposta amb seguretat abans d'acabar la seqüència o no per determinar de quin esquema es tracta.

9. La condició d'acabament

En un programa amb una seqüència amb més d'un element, la condició d'acabament és més complicada. Per exemple, si volem calcular la longitud màxima dels vectors 2D d'una seqüència i el sentinella és el vector (0,0), haurem de fer un programa com aquest:

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  float x, y, long, long_max;
  cin >> x >> y;
  long_max = sqrt( x*x + y*y );
  while (            ) {
    long = sqrt( x*x + y*y );
    if ( long > long_max ) long_max = long;
    cin >> x >> y;
  }
  cout << "Longitud màxima: " << long_max;
}
```

Pensa com posar la condició d'acabament (en blau) per tal que el programa acabi només quan veu que **`x`** i **`y`** són 0, però funciona bé amb la següent seqüència (demostra-ho!):

```
5.1 5.0
4.3 7.1
0.0 9.0
0.2 8.7
11.3 0.0
0.0 0.0
```

El problema, si t'hi fixes, és que hi ha vectors a on la **`x`** és 0.0 (però la **`y`** no) i d'altres a on la **`y`** és 0.0 (però la **`x`** no). El programa ha d'acabar només quan *tots dos* siguin 0.0.

10. El màxim de la mitjana

Suposa la següent seqüència:

```
mates 7.0
mates 4.5
mates 8.0
mates 5.0
fisica 10.0
fisica 8.0
fisica 3.0
fisica 6.0
informatica 7.0
informatica 6.0
informatica 10.0
informatica 8.5
FI 0.0
```

La seqüència és d'assignatures i amb cada una d'elles hi ha una nota. El sentinella és "FI 0.0", com pots veure, i s'ha d'anar amb compte perquè pot haver-hi una nota que sigui 0.0, però no sigui el final de la seqüència. La cosa important és que les notes de la mateixa assignatura són consecutives, la seqüència es defineix així. Fes, doncs, un programa que dóna com a resultat l'assignatura amb la nota mitjana màxima.

11. Fill the gaps

a) Donat el programa *fill_gaps_a.cpp* es demana que **implementeu el codi que falta** per tal de que faci el següent:
Es demana un valor parell entre 20 i 50 i un altre valor entre 80 i 100. El programa visualitza tots els parells que estan entre aquest límits i mostra la suma d'aquest parells i el total de valors visualitzats.
Falta fer 2 condicions (while) i el tractament de l'element.

b) Donat el programa *fill_gaps_b.cpp* es demana que **implementeu el codi que falta** per tal de que faci el següent:
Calculadora senzilla. Donat un menú, és demana una opció a l'usuari i es permet sumar, restar o multiplicar sobre el resultat previ. Per cada càlcul es va donant el resultat fins a aquell moment. El programa finalitza quan s'escull l'opció de sortir.
Falta fer 1 condició (while) i el tractament de l'element.

c) Donat el programa *fill_gaps_c.cpp* es demana que **corregiu els errors** de funcionament per tal de que faci el següent:
Determinar si un any es de traspàs o no. L'usuari introdueix l'any i el programa visualitza un missatge indicant si es de traspàs o no. Per cada valor es pregunta a l'usuari si desitja continuar, el programa finalitza quan l'usuari indiqui que no vol continuar. Al final el programa visualitza el nombre total d'anys de traspàs que s'han introduït.
(Un any és de traspàs si es divisible per 4, amb l'excepció dels anys de fi de segle que seran de traspàs només si son divisibles per 400.)
Feu un joc de proves adequat.

4) Donat el programa *fill_gaps_d.cpp* es demana que **corregiu els errors** de funcionament per tal de que faci el següent:

Dir si en una compra s'han comprat més de 1000 unitats d'algun producte, si no és així mostrar el total de la compra.
L'usuari introdueix el codi de producte, el preu del producte i les unitats comprades i es visualitza el total per aquell producte. El programa finalitza quan es troba algun producte amb més 1000 unitats o quan l'usuari introdueixi un codi de producte negatiu.
Feu un joc de proves adequat.

Altres exercicis:

1. Fora o dintre

Donat el centre d'una circumferència i un radi indicar si una sèrie de 200 punts del pla introduïts pel teclat pertanyen a l'interior o l'exterior de la circumferència.

2. Horitzontal o vertical

Implementeu un programa que permeti introduir una llista de 30 punts del pla (2 dimensions) i indiqui si dos punts consecutius formen una recta horitzontal (paral·lela al eix de les x) o una recta vertical (paral·lela al eix de les y). (Mireu l'exercici *Recordar l'anterior*)