MP 20-21 Tema 1
Fitxers – Serialització d'objectes
Sessió 10: Programació orientada a objectes

Exercici

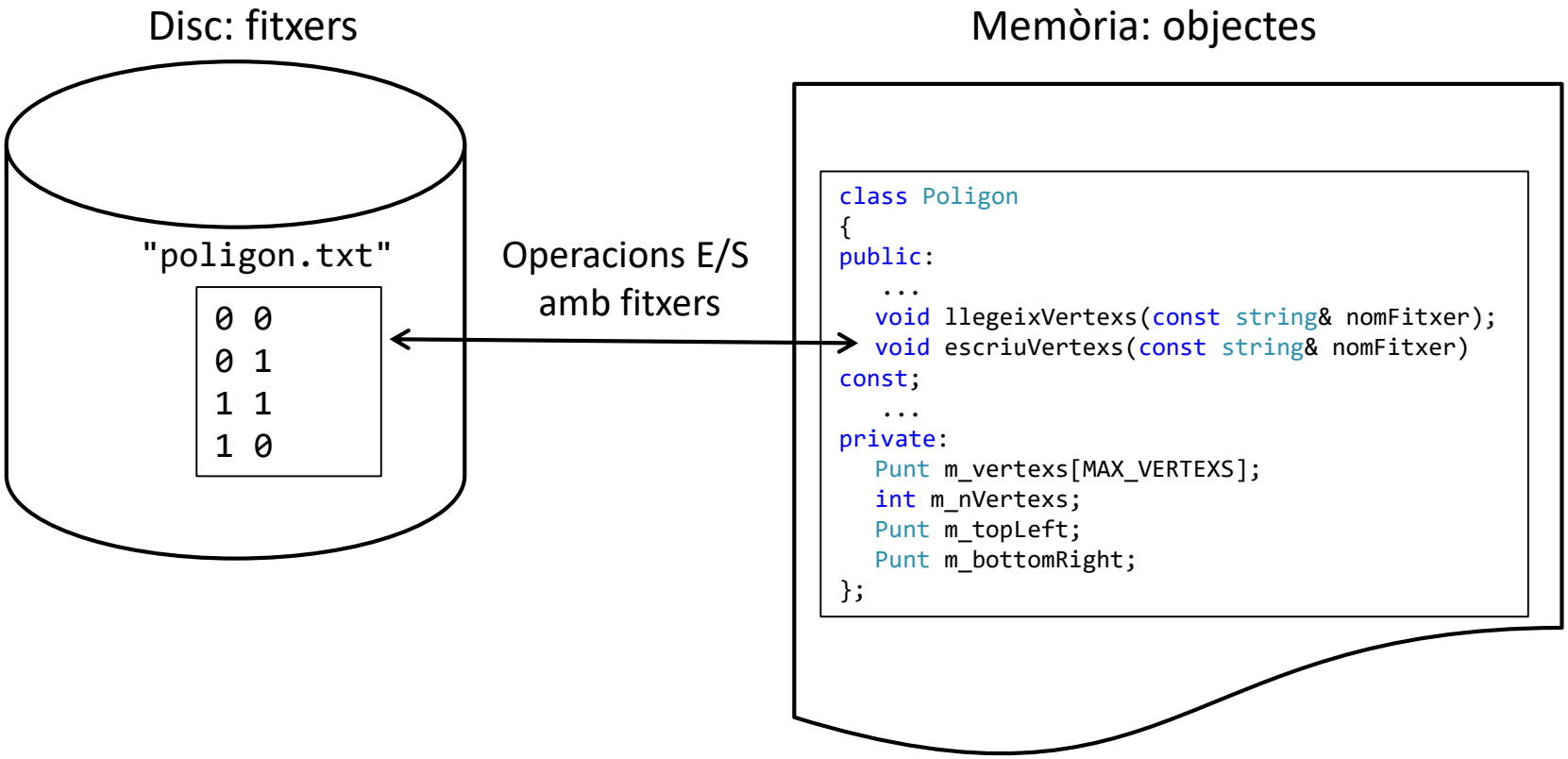
Recuperem la classes `Poligon` que hem utilitzat en exercicis anteriors:

- Afegir mètodes a la classe `Poligon` per llegir les dades dels vèrtexs d'un fitxer i per per escriure les dades de tots els vèrtexs a un fitxer

```
class Poligon
{
public:
   Poligon();
   ~Poligon() {}
   void llegeixVertexs(const string& nomFitxer);
   void escriuVertexs(const string& nomFitxer) const;
   void afegeixVertex(const Punt& pt);
   Punt getTopLeft() const;
   Punt getBottomRight() const;
   float calculaPerimetre() const;
private:
   Punt m_vertexs[MAX_VERTEXS];
   int m_nVertexs;
   Punt m_topLeft;
   Punt m_bottomRight;
};
```

Llegir/escriure dades de fitxers

- Afegir mètodes a la classe `Poligon` per llegir les dades dels vèrtexs d'un fitxer i per per escriure les dades de tots els vèrtexs a un fitxer

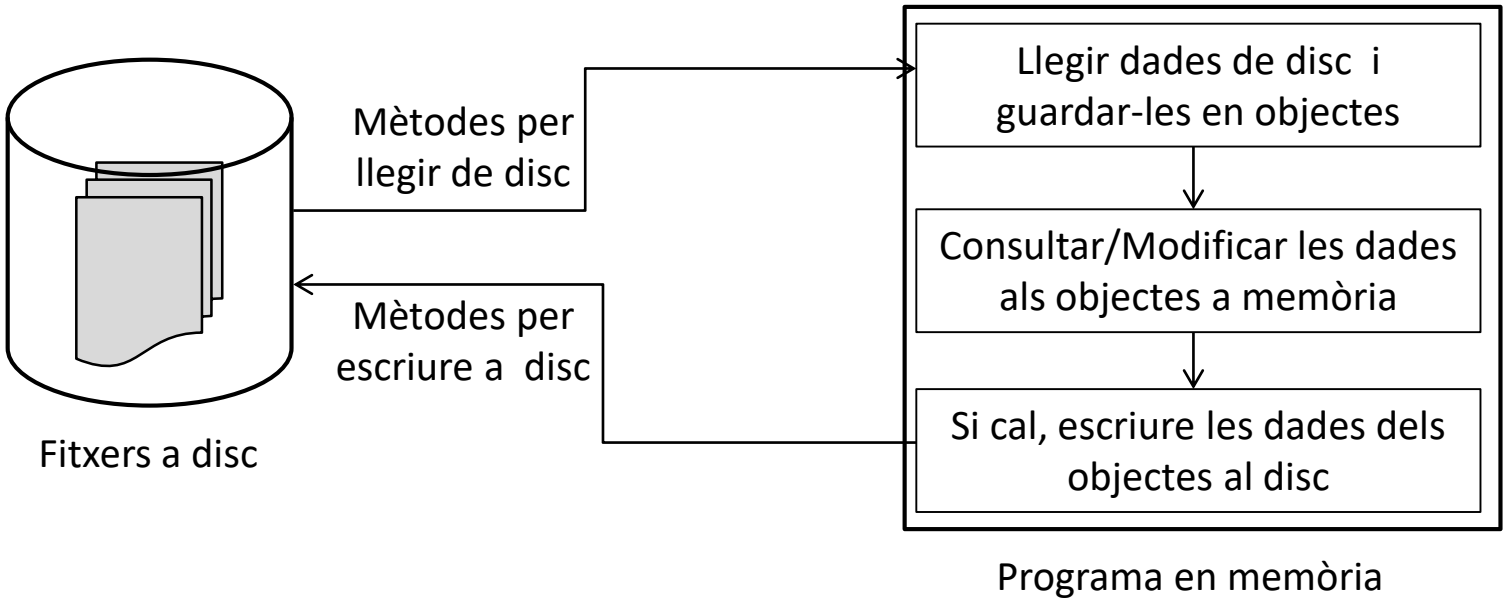


Els mètodes `llegeixVertexs/escriuVertexs` permetran gestionar el traspàs d'informació entre el fitxer i un objecte de la classe Poligon

Fitxers

- Permeten accedir a informació que tenim guardada al disc de forma permanent
- Utilització limitada: accés molt més lent que a memòria (variables del programa):
  - Utilitzar els fitxers només com a repositori per guardar de forma permanent la informació que ens interessa mantenir
  - De forma habitual treballarem amb les dades en memòria

Esquema habitual de treball amb la informació guardada al disc



Fitxers: classes `ifstream` i `ofstream`

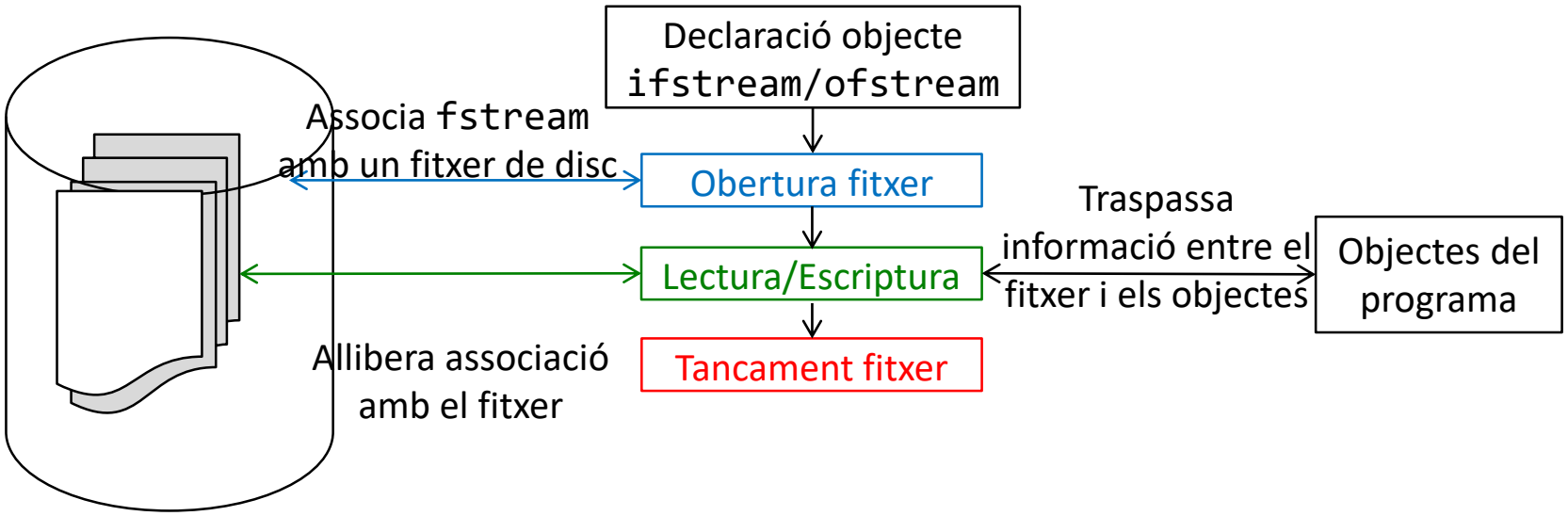
Accés als fitxers:

- Utilització de les classes **`ifstream`** (per llegir de fitxer) / **`ofstream`** (per escriure al fitxer)
  - Classes predefinides per accedir a fitxers de text en mode seqüencial
  - Contenen tots els mètodes i operacions bàsiques per accedir als fitxers
  - Funcionament molt similar a les classes `istream` i `ostream` que utilitzem per llegir de teclat amb `cin` i per escriure per pantalla amb `cout`
  - La principal diferència està en què abans de començar a llegir/escriure hem d'associar el canal de lectura (`ifstream`) o el canal de sortida (`ofstream`) amb el fitxer d'on volem llegir/escriure informació
  - Un cop associats amb el fitxer, la forma de llegir i escriure és la mateixa que s'utilitza amb els canals d'entrada/sortida per teclat (`cin`) i pantalla (`cout`)

Fitxers: classes `ifstream` i `ofstream`

Esquema bàsic d'accés als fitxers:

1. **Obertura del fitxer**: associa una variable `ifstream`/`ofstream` amb un fitxer del disc per llegir o per escriure
2. **Lectura/Escriptura** del fitxer: traspassa la informació del fitxer de disc a memòria (variables del programa) o a l'inrevés.
   - Accés seqüencial tant per llegir com per escriure de forma molt similar a la utilització dels canals estàndards `cin` i `cout`.
3. **Tancament del fitxer**: allibera l'associació de la variable `ifstream` / `ofstream` amb el fitxer de disc



Fitxers: classes `ifstream` i `ofstream`

Esquema bàsic d'accés als fitxers:

1. **Obertura del fitxer**: associa una variable `ifstream`/`ofstream` amb un fitxer del disc per llegir o per escriure

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

ifstream fitxer;
fitxer.open("nom_fitxer.txt");
```

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

ofstream fitxer;
fitxer.open("nom_fitxer.txt");
```

2. **Lectura/Escriptura** del fitxer: traspassa la informació del fitxer de disc a memòria (variables del programa) o a l'inrevés.
   - Accés seqüencial tant per llegir com per escriure de forma molt similar a la utilització dels canals estàndards `cin` i `cout`.

```
int x;
fitxer >> x;
```

```
int x = 10;
fitxer << x;
```

3. **Tancament del fitxer**: allibera l'associació de la variable `ifstream` / `ofstream` amb el fitxer de disc

```
fitxer.close();
```

```
fitxer.close();
```

Exercici

Recuperem la classes `Poligon` que hem utilitzat en exercicis anteriors:

- Afegir mètodes a la classe `Poligon` per llegir les dades dels vèrtexs d'un fitxer i per per escriure les dades de tots els vèrtexs a un fitxer

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

void Poligon::escriuVertexs(const string& nomFitxer) const
{
   ofstream fitxer;
   fitxer.open(nomFitxer);

   fitxer << m_nVertexs << endl;
   for (int i = 0; i < m_nVertexs; i++)
   {
      fitxer << m_vertexs[i].getX() << " " << m_vertexs[i].getY() << endl;
   }
   fitxer.close();
}
```

Declaració variable fitxer sortida

Obertura fitxer. Associació variable fitxer amb fitxer disc

Escriptura de dades al fitxer.

Tancament del fitxer

"poligon.txt"

```
4
0 0
0 1
1 1
1 0
```

Exercici

Recuperem la classes `Poligon` que hem utilitzat en exercicis anteriors:

- Afegir mètodes a la classe `Poligon` per llegir les dades dels vèrtexs d'un fitxer i per per escriure les dades de tots els vèrtexs a un fitxer

"poligon.txt"

```
4
0 0
0 1
1 1
1 0
```

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

void Poligon::llegeixVertexs(const string& nomFitxer)
{
   ifstream fitxer;
   fitxer.open(nomFitxer);

   int n;
   fitxer >> n;
   for (int i = 0; i < n; i++)
   {
      float x, y;
      fitxer >> x >> y;
      Punt pt(x, y);
      afegeixVertex(pt);
   }
   fitxer.close();
}
```

Declaració variable fitxer entrada

Obertura fitxer. Associació variable fitxer amb fitxer disc

Lectura de dades del fitxer.

Tancament del fitxer

Accés als fitxers

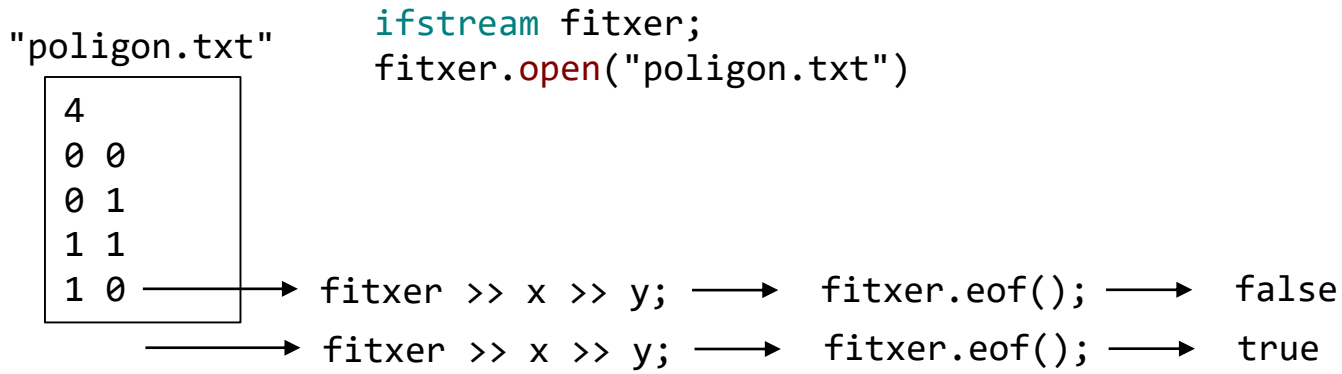
- Obertura dels fitxers:
  - Lectura:
    - El fitxer ha d'existir
    - Es comença a llegir sempre pel principi del fitxer
  - Escriptura:
    - Si el fitxer no existeix, el crea. Si el fitxer ja existeix, per defecte, esborra el contingut anterior i el torna a crear de nou.
    - Si volem conservar el contingut anterior, hem d'obrir el fitxer en mode "append": els nous valors s'escriuen al final del contingut previ.

```
ofstream fitxer;
fitxer.open("nom_fitxer.txt", ofstream::app);
```

- L'accés als fitxers és seqüencial:
  - Lectura: cada operació de lectura amb l'operador >> llegeix el següent valor del fitxer (valors separats per espais en blanc o salt de línia)
  - Escriptura: cada operació d'escriptura amb l'operador << escriu un valor al final del fitxer.

Accés als fitxers

- Quan llegim d'un fitxer, com podem saber si hem arribat al final?
  - `eof()`: mètode de la classe `ifstream` que retorna `true` si hem intentat llegir un cop hem arribat al final del fitxer.
    - Després de llegir correctament l'últim valor del fitxer, retorna `false`.
    - Només retorna `true` quan fem una lectura després de llegir l'últim valor del fitxer.



Lectura de fitxers

Afegir un mètode a la classe `Poligon` per llegir els vèrtexs d'un fitxer

```
#include <fstream>
using namespace std;
```
→ Inclusió dels fitxers de llibreria necessaris

```
void Poligon::llegeixVertexs(const string& nomFitxer)
{
   ifstream fitxer;
```
→ Declaració de l'objecte de la classe `ifstream` per poder llegir del fitxer

```
   fitxer.open(nomFitxer);
```
→ Obertura del fitxer. `nomFitxer` és un `string` que conté el nom del fitxer al que volem accedir

```
   if (fitxer.is_open())
```
→ Retorna `true` si el fitxer s'ha obert correctament

```
   {
      float x, y;
      fitxer >> x >> y;
```
→ Lectura de dades del fitxer. Sintaxi idèntica a lectura amb `cin`

```
      while (!fitxer.eof())
```
→ Retorna `true` si s'intenta llegir més enllà del final del fitxer. Primer llegim un valor i després comprovem final de fitxer

```
      {
         Punt pt(x, y);
         afegeixVertex(pt);
         fitxer >> x >> y;
      }
      fitxer.close();
```
→ Tancament del fitxer

```
   }
}
```

Lectura de fitxers

Esquema general de lectura de dades

```
// Inclusió fitxers llibreria
#include <fstream>
using namespace std;

ifstream fitxer;
// Variable per guardar el nom del fitxer
string nomFitxer = "nomDelFitxer.txt"
fitxer.open (nomFitxer);


if (fitxer.is_open())
{
     int numero;
     fitxer >> numero;


     while (!fitxer.eof())
     {

          <tractar_numero>
          fitxer >> numero;
     }
     fitxer.close();
}
```

Declaració variable fitxer entrada

Obertura fitxer. Associació variable fitxer amb fitxer disc

Comprovar si fitxer obert correctament

Lectura de dades del fitxer. Suposem que el fitxer conté números enters

Comprovar final del fitxer.S'activa quan s'intenta llegir després final. Primer llegir i després comprobar final

Tractar el valor llegit

Tancament del fitxer

Escriptura de fitxers

Afegir un mètode a la classe `Poligon` per llegir els vèrtexs d'un fitxer

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

→ Inclusió dels fitxers de llibreria necessaris

```
void Poligon::escriuVertexs(const string& nomFitxer) const
{
   ofstream fitxer;
```

→ Declaració de l'objecte de la classe `ofstream` per poder llegir del fitxer

```
   fitxer.open(nomFitxer);
```

→ Obertura del fitxer. `nomFitxer` és un `string` que conté el nom del fitxer al que volem accedir

```
   for (int i = 0; i < m_nVertexs; i++)
   {
      fitxer << m_vertexs[i].getX() << " " << m_vertexs[i].getY() << endl;
   }
```

→ Escriptura de dades al fitxer en el format que sigui necessari. Sintaxi idèntica a escriptura amb `cout`

```
   fitxer.close();
}
```

→ Tancament del fitxer

Lectura de fitxers

Esquema general d'escriptura de dades

```
// Inclusió fitxers llibreria
#include <fstream>
using namespace std;

ofstream fitxer;
// Variable per guardar el nom del fitxer
string nomFitxer = "nomDelFitxer.txt"
fitxer.open (nomFitxer);


if (fitxer.is_open())
{
     int numero;
     while (<hi_ha_valors_per_escriure>)
     {
          <obtenir_numero>


          fitxer << numero;
     }

     fitxer.close();
}
```

Declaració variable fitxer sortida

Obertura fitxer. Associació variable fitxer amb fitxer disc

Comprovar si fitxer obert correctament

Comprovar si queden valors per escriure

Obtenir el següent valor per escriure

Escriptura de dades al fitxer. Suposem que el fitxer conté números enters

Tancament del fitxer

Fitxers: sobrecàrrega operador >>

```
float x, y;
fitxer >> x >> y;

while (!fitxer.eof())
{
   Punt pt(x, y);
   afegeixVertex(pt);
   fitxer >> x >> y;
}
```

```
Punt pt;
fitxer >> pt;
while (!fitxer.eof())
{
   afegeixVertex(pt);
   fitxer >> pt;
}
```

L'operador >> de la classe `ifstream` es pot sobrecarregar per llegir de fitxers igual que es pot fer per llegir de teclat amb la classe `istream` i `cin`.

```
fitxer >> pt;
```

↓

```
operator>>(fitxer, pt)
```

↓

```
ifstream& operator>>(ifstream& input, Punt& pt)
{
   float x, y;
   input >> x >> y;
   pt.setX(x);
   pt.setY(y);
   return input;
}
```

Fitxers: sobrecàrrega operador <<

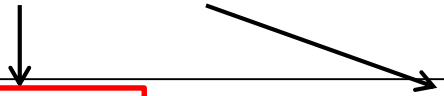
```
for (int i = 0; i < m_nVertexs; i++)
   fitxer << m_vertexs[i].getX() << " " << m_vertexs[i].getY() << endl;
```

```
for (int i = 0; i< m_nVertexs; i++)
   fitxer << m_vertexs[i];
```

L'operador << de la classe `ofstream` es pot sobrecarregar per escriure a fitxers igual que es pot fer per escriure a pantalla amb la classe `ostream` i `cout`.

```
fitxer << m_vèrtexs[i];
```

```
operator<<(fitxer, m_vertexs[i])
```



```
ofstream& operator<<(ofstream& output, const Punt& pt)
{
   output << "(" << pt.getX() << ", " << pt.getY() << ")";
   return output;
}
```

Exercici

A partir de la classe Data que ja hem fet servir en els exercicis anteriors:

1. Sobrecarregueu els operadors `<<` i `>>` de les classes `ifstream` i `ofstream` per poder llegir i escriure dates a fitxer.
   - Per llegir la data del fitxer podeu suposar que per cada data tindreu una línia del fitxer amb el dia, el mes i l'any separats per espais en blanc.
   - La data s'ha d'escriure en format DD/MM/YYYY
2. Implementeu una funció `ordenaDates` amb aquesta capçalera:

```
void ordenaDates(const string& nomFitxerIn, const string& nomFitxerOut);
```

La funció ha de llegir totes les dates que hi ha al fitxer `nomFitxerIn` (una data a cada línia del fitxer), ordenar-les de més petita a més gran (utilitzant l'operador `<` que ja té implementat la classe `Data`) i escriure-les ordenades al fitxer `nomFitxerOut`.

Per ordenar les dates, les podeu anar guardant en un array. Cada cop que llegiu una nova data del fitxer d'entrada, busqueu a quina posició de l'array s'ha de posar per mantenir l'ordre amb les que ja teniu, i la guardeu en aquesta posició desplaçant totes les dates posteriors de l'array una posició a la dreta. Al final, guardeu l'array ordenat al fitxer de sortida.

(*) En la implementació de la classe Data que us donem hi hem afegit l'operador d'assignació entre dates, de forma que podeu assignar directament dues dates entre si:

```
Data data1, data2;
data2 = data1;
```