Projecte de ESIN
Normativa i Enunciat

Tardor de 2020

Aquest document és llarg però és imprescindible que el llegiu íntegrament i amb deteniment, àdhuc si sou repetidors, ja que es donen les instruccions i normes que heu de seguir per a que el vostre projecte sigui avaluat positivament. El professorat de l'assignatura donarà per fet que tots els alumnes coneixen el contingut íntegre d'aquest document.

Continguts:

1 Normativa

1. Tal i com s'explica a la Guia Docent, per a assolir els objectius de l'assignatura es considera imprescindible el desenvolupament per part de l'estudiant d'un projecte que requereix algunes hores addicionals de treball personal, apart de les classes de laboratori, on es fa el desenvolupament dels altres exercicis pràctics que permeten familiaritzar-vos amb l'entorn de treball i el llenguatge de programació C++.

2. El projecte es realitzarà en equips de dos estudiants. Si un d'ells abandona, un dels integrants ho haurà de notificar amb la màxima promptitud via e-mail a `jesteve@cs.upc.edu` i, eventualment, continuar el projecte en solitari. D'altra banda, només es permetrà la formació d'equips individuals en casos excepcionals on sigui impossible reunir-se o comunicar-se amb altres estudiants, i s'haurà de justificar mitjançant algun tipus de document.

3. El suport que fareu servir per aquest projecte és el llenguatge de programació C++ (específicament el compilador GNU `g++-9.3.0`) sobre l'entorn Linux del STIC. Això no és obstacle per al desenvolupament previ en PCs o similars. De fet, existeixen compiladors de C++ per a tota classe de plataformes i hauria de ser senzill el trasllat des del vostre equip particular a l'entorn del STIC, especialment si treballeu amb GNU/Linux en el vostre PC.

   Atenció: Existeix la possibilitat de petites incompatibilitats entre alguns compiladors de C++. En tot cas és imprescindible que feu almenys una comprovació final que el programa desenvolupat en PC o similar funciona correctament en l'entorn Linux del STIC (us podeu connectar remotament al servidor `ubiwan.epsevg.upc.edu`).

4. El projecte serà avaluat mitjançant:

   - la seva execució en l'entorn Linux del STIC amb una sèrie de *jocs de prova* i
   - la correcció del disseny, implementació i documentació: les decisions de disseny i la seva justificació, l'eficiència dels algorismes i estructures de dades, la legibilitat, robustesa i estil de programació, etc. Tota la documentació ha d'acompanyar el codi; no heu de lliurar cap documentació en paper.

   Existeixen dos tipus de jocs de prova: públics i privats. Els jocs de prova públics per a que podeu provar el vostre projecte estaran a la vostra disposició amb antelació suficient al Campus Digital (`https://atenea.upc.edu`).

   La nota del projecte es calcula a partir de la nota d'execució (E) i la nota de disseny (D). La nota total és:

   $$P = 0.4E + 0.6D$$

   si ambdues notes parcials (E i D) són majors que 0; $P = 0$ si la nota de disseny és 0.

   El capítol G del *Manual de laboratori d'ESIN* descriu, entre altres coses, les situacions que originen una qualificació de 0 en el disseny (i per tant una qualificació de 0 del projecte). La nota d'execució (E) és 3 punts com a mínim si s'han superat els jocs

de prova públics; en cas contrari, la nota és 0. Els jocs de prova privats poden aportar fins a 7 punts més, en cas que s'hagin superat els jocs de prova públics.

5. La data límit del lliurament final és el 17 de gener de 2021 a les 23:59. Si un equip no ha lliurat el projecte llavors la nota serà 0. Al Campus Digital (`https://atenea.upc.edu`) es donaran tots els detalls sobre el procediment de lliurament del projecte.

6. No subestimeu el temps que haureu d'esmerçar a cadascun dels aspectes del projecte: disseny, codificació, depuració d'errors, proves, documentació, ...

2 Enunciat del projecte

El propòsit d'aquest projecte és desenvolupar un sistema de gestió automàtica d'una terminal de contenidors de càrrega.

Típicament, una terminal de contenidors de càrrega és una gran esplanada organitzada en N fileres de contenidors, cadascuna de les quals té M places. En el problema que ens ocupa les places són rectangulars, de 10 peus de llarg per 6 d'amplada i es situen una darrera l'altra formant una filera. Entre les fileres hi ha passadissos per permetre la circulació de les grues encarregades de col·locar i retirar els contenidors. Les N fileres constitueixen el que s'anomena *àrea d'emmagatzematge*. A més hi sol haver una *àrea d'espera i maniobres* on es poden emmagatzemar temporalment un petit nombre de contenidors (en aquest projecte, i per simplificar, no es limitarà la capacitat d'aquesta àrea).

Tots els contenidors tenen la mateixa amplada i alçada, però la seva longitud és diferent: n'hi ha de 10, 20 i 30 peus[1]. Això significa que un contenidor ocuparà l'equivalent a una, dues o tres places en funció de la seva longitud. Per altra banda, els contenidors es poden apilar uns sobre els altres, sempre i quan la totalitat de la seva base toqui el terra o contenidors en el "pis" inferior. Ateses les limitacions de pes que poden suportar els contenidors i a les limitacions de les grues de la terminal, només es poden apilar fins un màxim de H pisos. Habitualment, $H = 3$ i mai serà més gran que una certa constant petita `HMAX`.

Les fileres de la terminal es numeren de 0 a $N - 1$, i les places dins d'una filera es numeren de 0 a $M - 1$. Els nivells o pisos es numeren de 0 a $H - 1$. Una *ubicació* és una terna d'enters $\langle i, j, k \rangle$ que designen, respectivament, un número de filera, un número de plaça i un pis. La ubicació d'un contenidor és la terna corresponent a la plaça ocupada pel contenidor amb menor j (cal recordar que els contenidors de 20 i 30 peus ocupen 2 i 3 places, respectivament). S'usa el mateix conveni per la ubicació d'un forat o grup de places consecutives lliures.

Quan un contenidor nou arriba a la terminal, el sistema ha de trobar-li un lloc apropiat en l'àrea d'emmagatzematge, automàticament. Si existeixen diversos espais lliures en els que es pot posar el nou contenidor, necessitarem una estratègia per decidir quin d'ells triar, intentant que:

1. sigui senzilla d'implementar i l'algorisme corresponent sigui eficient en temps d'execució i en espai de memòria;
2. aprofiti de la millor manera possible l'espai de la terminal, minimitzant la *fragmentació*[2].

[1] En realitat, la situació és més complexa i la variabilitat en les mides és més gran, però com ja hem dit treballarem amb un model simplificat.

[2] S'anomena *fragmentació* al fenomen pel qual no es pot ubicar un contenidor en l'àrea d'emmagatze-

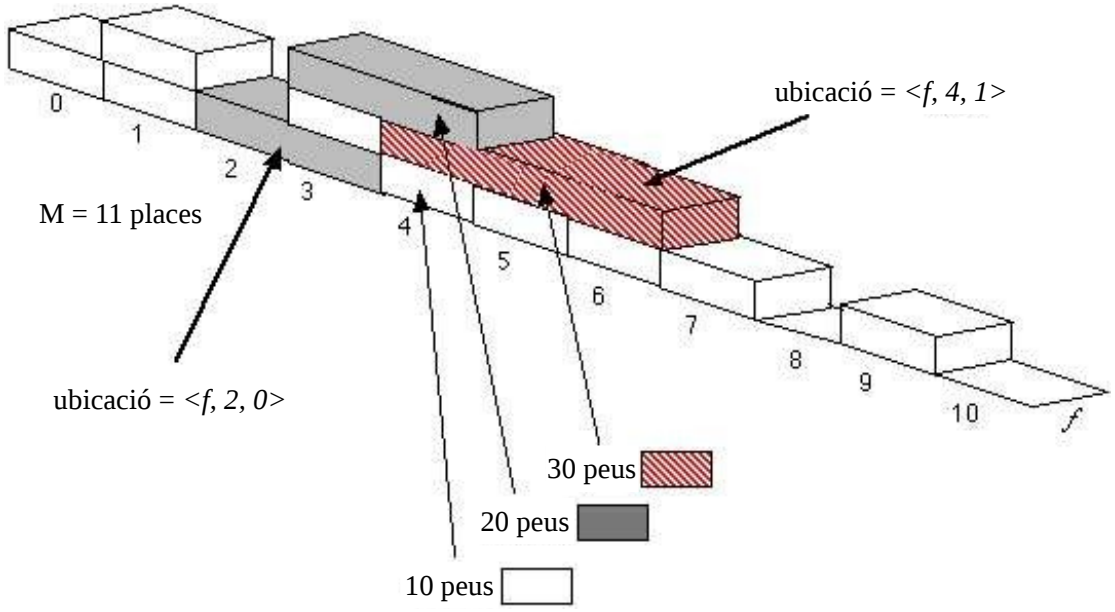

Figura 1: Esquema d'una filera de la terminal de contenidors

A més d'establir quin espai lliure escollir per un nou contenidor, una estratègia pot intentar crear espai lliure per un nou contenidor *reorganitzant* els ja existents. Si, per qualsevol motiu, un contenidor no pot ser col·locat a l'àrea d'emmagatzematge, llavors es col·loca en l'àrea d'espera.

En aquest projecte haureu d'implementar dues estratègies per insercions i retirades de contenidors en la terminal. Una d'elles s'anomena `FIRST_FIT` i es descriu amb detall en la subsecció 2.1. L'altra estratègia serà de vostra invenció i s'anomena `LLIURE`. Els principis que heu de seguir per aquesta darrera estratègia es descriuen en la subsecció 2.2. .

> Per poder comparar els costos espacials i temporals de diferents alternatives suposeu que la terminal està ocupada entre el 40 i el 60 per cent de la seva capacitat.

2.1 L'estratègia FIRST_FIT

L'estratègia FIRST_FIT estableix:

1. quin forat triar per un contenidor si hi ha diverses possibilitats;
2. com gestionar l'àrea d'espera.

matge havent però espai per això.

Durant una inserció, si existeixen diversos llocs lliures possibles pel nou contenidor, `FIRST_FIT` tria el "primer" d'ells en un hipotètic recorregut per la terminal de filera en filera, plaça a plaça, és a dir, preferentment s'usarà el forat amb el número de filera més baix i en cas d'empat, el forat amb el número de plaça més baix (no pot haver dos forats amb igual nombre de filera i de plaça). Aquesta característica concreta de l'algorisme és la que dóna nom a l'estratègia.

Si no hi hagués cap forat capaç d'albergar al contenidor, `FIRST_FIT` no pretén realitzar cap reorganització dels contenidors de l'àrea d'emmagatzematge, i es limita a posar el nou contenidor en l'àrea d'espera.

`FIRST_FIT` gestiona l'àrea d'espera com una llista de contenidors on els últims contenidors inserits a la llista són els que s'intenten recol·locar en primer lloc. Cada vegada que l'operació `insereix_contenidor` té èxit aconseguint col·locar el nou contenidor en l'àrea d'emmagatzematge es recorrerà l'àrea d'espera, començant pel que es va inserir en últim lloc en l'àrea d'espera, en busca d'un contenidor que pugui ser mogut a l'àrea d'emmagatzematge. Si la cerca té èxit s'utilitza la regla `FIRST_FIT` per moure el contenidor en qüestió des de l'àrea d'espera a la d'emmagatzematge. Aquest procés es repeteix fins que l'àrea d'espera s'ha examinat completament sense que hi hagi cap contenidor susceptible de ser mogut a l'àrea d'emmagatzematge.

En quant a l'operació de `retira_contenidor`, l'estratègia `FIRST_FIT` es limita a treure els contenidors que hi hagi per damunt del contenidor a retirar (si n'hi ha), posant-los a l'àrea d'espera. Després es recorre la llista intentant reubicar contenidors de l'àrea d'espera a la d'emmagatzematge, seguint un procés idèntic al que s'ha explicat anteriorment per la inserció.

Com exemple concret de funcionament, suposem una terminal (molt petita) amb $N = 1$, $M = 6$ i $H = 3$, inicialment buida. Usarem la nomenclatura `i` m n per indicar que s'insereix un contenidor amb matrícula m i $10 \cdot n$ peus, i `r` m per indicar que s'ha de retirar el contenidor de matrícula m. Suposem que s'han de processar les següents "peticions":

```
i A 3, i B 2, i C 3, i D 1, i E 1, i F 3, i G 1, i H 2, r A
```

Just abans de retirar el contenidor `A` l'estat de la terminal seria el següent:

```
espera: -   fragmentacio: 1   ops_grua: 8

pis 2  D E G H H
pis 1  B B F F F
pis 0  A A A C C C
```

Per retirar el contenidor `A` es mouen a l'àrea d'espera els contenidors `D`, `E`, `B`, `G`, `H` i `F`, per aquest ordre. Observeu que el contenidor `H` no queda damunt d'`A`, però ha de

moure's a l'àrea d'espera, per poder moure F. Observeu també que un cop retirats D i E es retira B abans que G ja que ambdós estan lliures en aquest moment però B té número de plaça més baix que G. Un cop retirat A es col·loquen tots els contenidors de l'àrea d'espera de nou a l'àrea d'emmagatzematge.

Si processem a continuació aquestes altres "peticions":

```
r E, r C, i I 1, i J 3, i K 2
```

l'estat final de la terminal després de processar aquestes altres peticions és:

```
espera: -   fragmentacio: 1   ops_grua: 30

pis 2  G D J J J
pis 1  H H I K K
pis 0  F F F B B _
```

Cal destacar que a l'inserir J no hi ha cap lloc per col·locar-lo de manera que queda a l'àrea d'espera; però després s'insereix K i llavors es pot col·locar J a sobre.

2.2 L'estratègia LLIURE

Els algorismes d'inserció i eliminació de contenidors corresponents a l'estratègia LLIURE són de la vostra elecció. Naturalment, tenen que ser diferents dels definits per FIRST_FIT. La vostra estratègia ha d'especificar quin lloc lliure triar si hi ha més d'una possibilitat, i en quin ordre es "processa" l'àrea d'espera (no ha de ser necessàriament en ordre invers del d'arribada, que és l'ordre que s'utilitza a l'estratègia FIRST_FIT).

La vostra estratègia també pot aplicar *reorganitzacions*, movent contenidors des de l'àrea d'emmagatzematge a l'àrea d'espera per fer lloc a un nou contenidor i després tornar a col·locar els contenidors moguts des de l'àrea d'espera a l'àrea d'emmagatzematge, evitant que el contenidor nou hagi de quedar a l'àrea d'espera per problemes de fragmentació. Tingueu en compte, però, que els algorismes de reorganització poden arribar a ser molt complicats, el seu cost en temps i espai ser molt elevat i necessitar moltes operacions de grua. A més, encara que el nombre de contenidors a l'àrea d'espera no estigui limitat en aquest projecte, és obvi que moure tots els contenidors de la terminal o tots els contenidors d'una filera a l'àrea d'espera per realitzar una reorganització no és factible en una situació real. Un algorisme de reorganització, si s'usa, mai hauria de moure més que uns pocs contenidors des de l'àrea d'emmagatzematge a l'àrea d'espera.

Observeu que, per exemple, l'estratègia FIRST_FIT **mai** reorganitza els contenidors, encara que procura que la fragmentació no sigui excessiva. Per exemple, evita col·locar contenidors en el "centre" d'un forat.

Finalment, és important adonar-se que totes les estratègies són *on-line*, és a dir, totes les decisions que es prenen, tant en les insercions com en les eliminacions, només es poden basar en la informació disponible en un instant sobre l'estat de la terminal i no sobre futures (i desconegudes) insercions i eliminacions.

Tres són els aspectes bàsics que mesuren la qualitat d'una estratègia:

- la seva senzillesa conceptual;
- el cost en temps i espai de l'algorisme que la implementa;
- i els valors "mitjans" dels paràmetres de qualitat: fragmentació, longitud de l'àrea d'espera i nombre d'operacions de grua.

Aquests tres aspectes (senzillesa, cost, paràmetres de qualitat) no necessàriament s'esmenten per ordre de importància.

3 Disseny modular

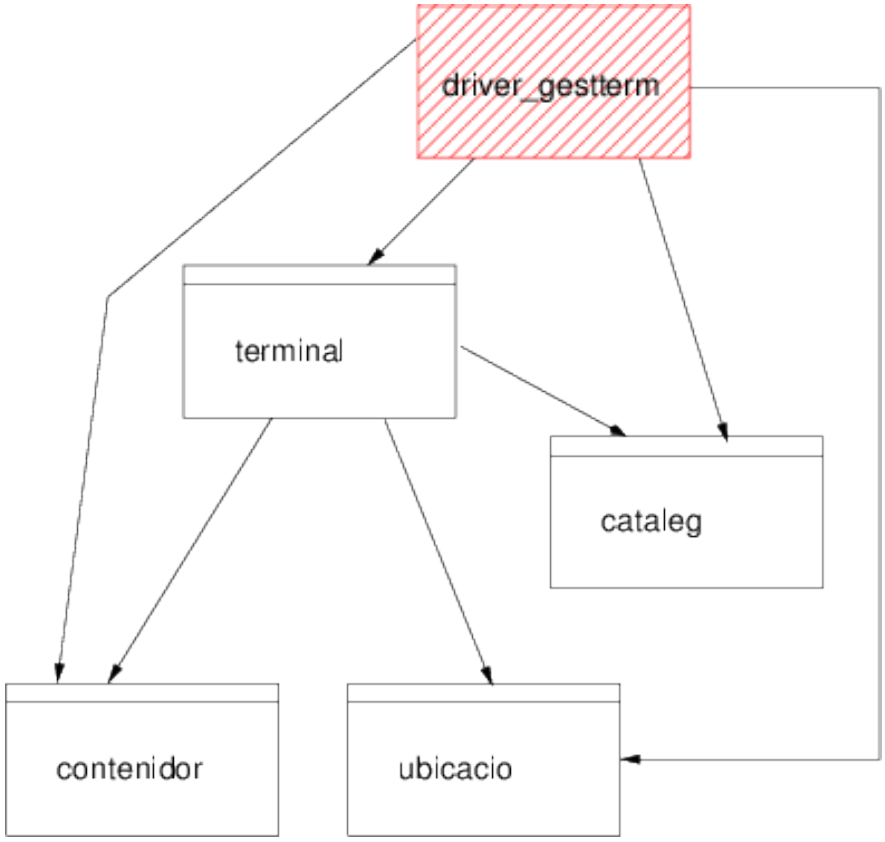

Figura 2: Disseny modular del projecte.

GestTerm consta d'una classe anomenada terminal que és l'encarregada de la gestió del terminal. Per tal de realitzar aquesta tasca, aquesta classe usa aquestes altres classes: ubicacio, contenidor i cataleg.

En aquest projecte la vostra tasca és la de dissenyar i implementar les classes contenidor, ubicacio, cataleg i terminal.

S'han omès d'aquest diagrama (figura 2) la classe error i el mòdul util de la biblioteca `libesin` per claredat, ja que moltes classes i mòduls del diagrama usen aquests mòduls. La classe error està documentada en el *Manual de laboratori d'ESIN*, i el mòdul util està documentat on-line en el fitxer `esin/util`. En algunes classes d'aquest projecte s'usa també la classe string i la list de la biblioteca estàndard de C++. Aquestes relacions d'ús i les classes en qüestió tampoc es mostren a la figura.

També tindreu a la vostra disposició, més endavant, els jocs de proves públics i el mòdul `driver_gestterm`. Aquest mòdul usa a totes les classes que heu d'implementar i permet invocar cada una de les operacions dels diferents mòduls i classes. Amb el driver podeu provar, de manera exhaustiva, les diferents classes.

IMPORTANT: Recordeu que no es pot utilitzar cap classe d'una biblioteca externa en les vostres classes, exceptuant si en aquesta documentació s'indica el contrari.

En totes les classes s'han d'implementar els mètodes de construcció per còpia, assignació i destrucció davant la possibilitat que useu memòria dinàmica en la classe en qüestió. Si no es fa ús de memòria dinàmica, la implementació d'aquests tres mètodes és molt senzilla doncs n'hi haurà prou amb imitar el comportament del que serien els corresponents mètodes d'ofici (el destructor no fa "res", i els altres fan còpies atribut per atribut).

Així mateix us proporcionem tots els fitxers capçalera (`.hpp`) d'aquest disseny modular. No podeu crear els vostres propis fitxers capçalera ni modificar de cap manera els que us donem. Tingueu present que heu de respectar escrupolosament l'especificació de cada classe que apareix en el corresponent `.hpp`.

ATENCIÓ: És important que un cop implementat cadascuna de les classes, les sotmeteu als vostres jocs de proves. A més, també és fonamental dissenyar amb força detall la representació de cada classe i els seus algorismes sobre paper, i prendre notes de tots els passos seguits abans de començar a codificar. Aquesta informació serà vital de cara a la preparació de la documentació final.

En resum, en aquest projecte les tasques que heu de realitzar i l'ordre que heu de seguir és el següent:

- Implementar la classe ubicacio
- Implementar la classe contenidor
- Implementar la classe cataleg
- Implementar la classe terminal

A l'hora de triar els algorismes i/o les estructures de dades per la representació d'una classe seguiu aquestes dues regles d'or:

1a. "Davant algorismes de costos temporals asimptòtics idèntics escolliu sempre el més senzill."

2a. "No usar exclusivament el criteri del cas pitjor"

4 La classe ubicacio

Una ubicació consisteix en una terna de números $\langle i, j, k\rangle$ corresponents a un número de filera, número de plaça i pis. Hi ha valors del tipus ubicacio que tenen significats especials (veure la secció 7): $\langle -1, 0, 0\rangle$ s'usa per indicar que el contenidor corresponent està a l'àrea d'espera i $\langle -1, -1, -1\rangle$ per indicar que un contenidor no està a la terminal.

Implementació: La representació d'aquesta classe es trobarà en el fitxer `ubicacio.rep` i la implementació en el fitxer `ubicacio.cpp`. La documentació haurà de ser mínima, donada la senzillesa de les dades i dels mètodes.

```
#ifndef _UBICACIO_HPP
#define _UBICACIO_HPP

#include <esin/error>
#include <esin/util>

class ubicacio {
public:

  /* Constructora. Crea la ubicació <i, j, k>. Genera un error si
    <i, j, k> no pertany a {<u, v, w> | u >= 0 ^ v >= 0 ^ w >= 0}
     o a {<-1, 0, 0>,  <-1,-1,-1>}. */
  ubicacio(int i, int j, int k) throw(error);

  /* Constructora per còpia, assignació i destructora. */
  ubicacio(const ubicacio& u) throw(error);
  ubicacio& operator=(const ubicacio& u) throw(error);
  ~ubicacio() throw();

  /* Consultors. Retornen respectivament el primer, segon i tercer
     component de la ubicació. */
  int filera() const throw();
  int placa() const throw();
  int pis() const throw();

  /* Operadors de comparació.
     L'operador d'igualtat retorna cert si i només si les dues ubicacions
     tenen la mateixa filera, plaça i pis. L'operador menor retorna cert si
     i només si la filera del paràmetre implícit és més petit que la
     d'u, o si les dues fileres són iguals i la plaça del paràmetre
     implícit és més petita que la d'u, o si les fileres i les places
     coincideixen i el pis del paràmetre implícit és més petit que el d'u.
     La resta d'operadors es defineixen consistentment respecte <. */
  bool operator==(const ubicacio &u) const throw();
  bool operator!=(const ubicacio &u) const throw();
  bool operator<(const ubicacio &u) const throw();
  bool operator<=(const ubicacio &u) const throw();
```

```
  bool operator>(const ubicacio &u) const throw();
  bool operator>=(const ubicacio &u) const throw();

  /* Gestió d'errors */
  static const int UbicacioIncorrecta = 10;

private:
  #include "ubicacio.rep"
};
#endif
```

5 La clase contenidor

Tot objecte de la classe contenidor té una matrícula o identificador format per una seqüència de lletres majúscula i dígits (p.e. AX02B7Z8) i una longitud (10, 20 o 30 peus). En una aplicació real es disposaria de més informació per cada contenidor: data d'entrada a la terminal, data prevista de sortida, descripció de la càrrega, pes, dades d'identificació del propietari, dades de procedència, model del contenidor, etc., però en aquest projecte prescindirem d'ella.

Decisions sobre les dades:

- Les matrícules es representaran mitjançant `string`s. La longitud de l'`string` no està limitada, però una matrícula no serà vàlida si l'`string` que la representa conté caràcters que no siguin lletres majúscules o dígits, o és l'`string` buit. Totes les lletres que formen la matrícula són lletres majúscules de l'A a la Z, excloses la Ñ, la Ç i les vocals accentuades. [3].
- La longitud del contenidor es representa mitjançant un enter positiu (`unsigned int`).

Implementació: La representació d'aquesta classe es trobarà en el fitxer `contenidor.rep` i la implementació en el fitxer `contenidor.cpp`. La documentació haurà de ser mínima, donada la senzillesa de les dades i dels mètodes.

```
#ifndef _CONTENIDOR_HPP
#define _CONTENIDOR_HPP

#include <string>
#include <esin/error>
#include <esin/util>

using std::string;
using util::nat;

class contenidor {
public:

  /* Constructora. Crea un contenidor amb matrícula m i longitud l.
     Es produeix un error si m no és una seqüència de un o més caràcters,
     formada exclusivament per lletres majúscules i dígits,
     o si l no pertany a {10, 20, 30} */
  contenidor(const string &m, nat l) throw(error);

  /* Constructora per còpia, assignació i destructora. */
  contenidor(const contenidor &u) throw(error);
```

[3] Les matrícules només contenen caràcters que els seus codis ASCII estan entre 65 ('A') i 90 ('Z') o siguin dígits

```
  contenidor& operator=(const contenidor &u) throw(error);
  ~contenidor() throw();

  /* Consultors. Retornen respectivament la longitud i la matrícula del
     contenidor. */
  nat longitud() const throw();
  string matricula() const throw();

  /* Operadors de comparació. L'operador d'igualtat retorna cert si i
     només si els dos contenidors contenen la mateixa matrícula i longitud.
     L'operador menor retorna cert si i només si la matrícula del
     paràmetre implícit es més petit en ordre alfabètic que la c o si les
     dues matrícules són iguals i la longitud del paràmetre implícit és més
     petita que la de c. La resta d'operadors es defineixen consistentment
     respecte a <. */
  bool operator==(const contenidor &c) const throw();
  bool operator!=(const contenidor &c) const throw();
  bool operator<(const contenidor &c) const throw();
  bool operator<=(const contenidor &c) const throw();
  bool operator>(const contenidor &c) const throw();
  bool operator>=(const contenidor &c) const throw();

  /* Gestió d'errors */
  static const int MatriculaIncorrecta = 20;
  static const int LongitudIncorrecta = 21;

private:
  #include "contenidor.rep"
};
#endif
```

6 La classe cataleg

Un objecte de la clase cataleg representa una col·lecció de parells $\langle clau, valor \rangle$ en la que no hi ha dos parells amb igual clau.

En el mètode constructor d'aquesta classe s'indica el nombre d'elements que com a màxim s'inseriran en el catàleg. En qualsevol cas el catàleg ha de permetre emmagatzemar elements més enllà del màxim indicat.

Decisions sobre les dades:

- Les claus són *strings* no buits.
- El tipus Valor ha de disposar del constructor per còpia i de l'operador assignació.

Implementació: La representació d'aquesta classe es trobarà en el fitxer `cataleg.rep` i la implementació en el fitxer `cataleg.t`.

Observacions: No es pot emprar cap classe de la biblioteca estàndard de C++, exceptuant la classe string per representar les claus.

```
#ifndef _CATALEG_HPP
#define _CATALEG_HPP
#include <string>
#include <esin/error>
#include <esin/util>

using std::string;
using util::nat;

template <typename Valor>
class cataleg {
public:

  /* Constructora. Crea un catàleg buit on numelems és el nombre
     aproximat d'elements que com a màxim s'inseriran al catàleg. */
  explicit cataleg(nat numelems) throw(error);

  /* Constructora per còpia, assignació i destructora. */
  cataleg(const cataleg& c) throw(error);
  cataleg& operator=(const cataleg& c) throw(error);
  ~cataleg() throw();

  /* Mètode modificador. Insereix el parell <clau, valor> indicat.
     En cas que la clau k ja existeixi en el catàleg actualitza el valor
     associat. Genera un error en cas que la clau sigui l'string buit. */
  void assig(const string &k, const Valor &v) throw(error);
```

```
  /* Elimina del catàleg el parell que té com a clau k.
     En cas que la clau k no existeixi en el catàleg genera un error. */
  void elimina(const string &k) throw(error);

  /* Retorna true si i només si la clau k existeix dins del catàleg; false
     en cas contrari. */
  bool existeix(const string &k) const throw();

  /* Retorna el valor associat a la clau k; si no existeix cap parell amb
     clau k llavors genera un error. Exemple:
       cataleg<int> ct;
       ...
       int n = ct["dia"]; */
  Valor operator[](const string &k) const throw(error);

  /* Retorna el nombre d'elements que s'han inserit en el catàleg
     fins aquest moment. */
  nat quants() const throw();

  /* Gestió d'errors */
  static const int ClauStringBuit = 30;
  static const int ClauInexistent = 31;

private:
  #include "cataleg.rep"
};
#include "cataleg.t"
#endif
```

7 La classe terminal

Aquesta classe ofereix dues operacions "principalment": `insereix_contenidor` i `retira_contenidor`, a més d'un cert nombre de mètodes auxiliars i el mètode constructor. La primera d'aquestes operacions s'utilitza quan un nou contenidor arriba a la terminal i la segona quan un contenidor abandona la terminal.

Per retirar un contenidor *c* (que està a l'àrea d'emmagatzematge) s'han de moure a l'àrea d'espera tots els contenidors que estiguin sobre el contenidor en qüestió, eliminar el contenidor *c* de la terminal i intentar recol·locar contenidors de l'àrea d'espera a l'àrea d'emmagatzematge. Si el contenidor que es vol retirar està a l'àrea d'espera, s'elimina de manera immediata.

Decisions sobre les dades:

- `HMAX` és una constant.
- Les operacions `on` i `longitud` reben un `string` qualsevol com a paràmetre.

Implementació: La representació d'aquesta classe es trobarà en el fitxer `terminal.rep` i la implementació en el fitxer `terminal.cpp`.

Observacions: Es pot emprar la classe list de la biblioteca estàndard de C++, però únicament per representar i gestionar l'àrea d'espera del terminal i implementar el mètode `area_espera`.

Atès que les operacions `insereix_contenidor` i `retira_contenidor` poden involucrar el moviment de varis contenidors des de l'àrea d'espera a l'àrea d'emmagatzematge, resultaria ineficient o molt complicat garantir que la o les estructures de dades que representen la terminal quedin en el seu estat original si es produís un error de memòria dinàmica durant un pas intermedi de l'execució. Per aquesta raó, i excepcionalment, la vostra implementació d'aquests dos mètodes no tindrà que deixar necessàriament inalterada l'estructura de dades en el cas que es produeixi un error de manca de memòria dinàmica. Si es produeix algun altre tipus d'error, la regla sí que s'aplica, ja que aquests altres errors poden ser fàcilment verificats abans de procedir a la modificació de l'estructura.

```
#ifndef _TERMINAL_HPP
#define _TERMINAL_HPP
#include <list>
#include <string>
#include <esin/error>
#include <esin/util>

#include "cataleg.hpp"
#include "contenidor.hpp"
```

```
#include "ubicacio.hpp"

using std::string;
using std::list;
using util::nat;

class terminal {
public:

  static const nat HMAX = 7;

  enum estrategia { FIRST_FIT, LLIURE };

  /* Constructora. Crea una terminal buida amb n fileres de m places
     cadascuna, i una alçada màxima d'apilament h; a més fixa l'estratègia
     d'inserció i retirada dels contenidors respecte el paràmetre st.
     Genera un error si n=0, m=0, h=0, h > HMAX o
     st no pertany a {FIRST_FIT, LLIURE}. */
  terminal(nat n, nat m, nat h, estrategia st) throw(error);

  /* Constructora per còpia, assignació i destructora. */
  terminal(const terminal& b) throw(error);
  terminal& operator=(const terminal& b) throw(error);
  ~terminal() throw();

  /* Col·loca el contenidor c en l'àrea d'emmagatzematge de la terminal o
     en l'àrea d'espera si no troba lloc en l'àrea d'emmagatzematge usant
     l'estratègia prefixada en el moment de crear la terminal. Si el
     contenidor c es col·loca en l'àrea d'emmagatzematge pot succeir que
     un o més contenidors de l'àrea d'espera puguin ser moguts a l'àrea
     d'emmagatzematge.
     En aquest cas es mouran els contenidors de l'àrea d'espera a l'àrea
     d'emmagatzematge seguint l'ordre que indiqui l'estratègia que s'està
     usant. Finalment, genera un error si ja existís a la terminal un
     contenidor amb una matrícula idèntica que la del contenidor c. */
  void insereix_contenidor(const contenidor &c) throw(error);

  /* Retira de la terminal el contenidor c la matrícula del qual és igual
     a m. Aquest contenidor pot estar a l'àrea d'emmagatzematge o a l'àrea
     d'espera. Si el contenidor estigués a l'àrea d'emmagatzematge llavors
     s'hauran de moure a l'àrea d'espera tots els contenidors que siguin
     necessaris per netejar la part superior de c, s'hauran de retirar
     possiblement diversos contenidors, començant pel contenidor sense cap
     altre a sobre amb el número de plaça més baix (més a l'esquerra) i així
     successivament (veure exemple amb detall a la subsecció Estratègia
     FIRST_FIT). Un cop s'hagi eliminat el contenidor indicat, s'intenta
     moure contenidors de l'àrea d'espera a l'àrea d'emmagatzematge, seguint
     l'ordre que indiqui l'estratègia que s'està usant. Genera un error si a
```

```
   la terminal no hi ha cap contenidor la matrícula del qual sigui igual a m. */
void retira_contenidor(const string &m) throw(error);

/* Retorna la ubicació <i, j, k> del contenidor la matrícula del qual és
   igual a m si el contenidor està a l'àrea d'emmagatzematge de la terminal,
   la ubicació <-1, 0, 0> si el contenidor està a l'àrea d'espera, i la
   ubicació <-1, -1, -1> si no existeix cap contenidor que tingui una
   matrícula igual a m.
   Cal recordar que si un contenidor té més de 10 peus, la seva ubicació
   correspon a la plaça que tingui el número de plaça més petit. */
ubicacio on(const string &m) const throw();

/* Retorna la longitud del contenidor la matrícula del qual és igual
   a m. Genera un error si no existeix un contenidor a la terminal
   la matrícula del qual sigui igual a m. */
nat longitud(const string &m) const throw(error);

/* Retorna la matrícula del contenidor que ocupa la ubicació u = <i, j, k>
   o la cadena buida si la ubicació està buida. Genera un error si
   i < 0, i >= n, j < 0, j >= m, k < 0 o k >= h, o sigui si <i, j, k> no
   identifica una ubicació vàlida de l'àrea d'emmagatzematge. Cal observar
   que si m, obtinguda amb t.contenidor_ocupa(u, m), és una matrícula (no
   la cadena buida) pot succeir que u != t.on(m), ja que un contenidor pot
   ocupar diverses places i la seva ubicació es correspon amb la de la
   plaça ocupada amb número de plaça més baix. */
void contenidor_ocupa(const ubicacio &u, string &m) const throw(error);

/* Retorna el nombre de places de la terminal que en aquest instant
   només hi cabrien un contenidor de 10 peus, però no un de més llarg.
   Per exemple, la filera de la figura 1 de l'enunciat contribuirà amb
   7 unitats a la fragmentació total (corresponen a les ubicacions
   <f, 0, 1>, <f, 1, 2>, <f, 2, 1>, <f, 7, 1>, <f, 8, 0>, <f, 9, 1> i
   <f, 10, 0>). */
nat fragmentacio() const throw();

/* Retorna el número d'operacions de grua realitzades des del moment
   de creació de la terminal.
   Es requereix d'una operació de grua per moure un contenidor
   des de l'àrea d'espera a l'àrea d'emmagatzematge o viceversa.
   També es requereix d'una operació de grua per inserir o
   retirar directament un contenidor de l'àrea d'emmagatzematge.
   En canvi no requereix cap operació de grua inserir o
   retirar directament un contenidor de l'àrea d'espera. */
nat ops_grua() const throw();

/* Retorna la llista de les matrícules de tots els contenidors
   de l'àrea d'espera de la terminal, en ordre alfabètic creixent. */
void area_espera(list<string> &l) const throw();
```

```
  /* Retorna el número de fileres de la terminal. */
  nat num_fileres() const throw();

  /* Retorna el número de places per filera de la terminal. */
  nat num_places() const throw();

  /* Retorna l'alçada màxima d'apilament de la terminal. */
  nat num_pisos() const throw();

  /* Retorna l'estratègia d'inserció i retirada de contenidors de
     la terminal. */
  estrategia quina_estrategia() const throw();

  /* Gestió d'errors */
  static const int NumFileresIncorr = 40;
  static const int NumPlacesIncorr  = 41;
  static const int AlcadaMaxIncorr  = 42;
  static const int EstrategiaIncorr = 43;
  static const int MatriculaDuplicada  = 44;
  static const int MatriculaInexistent = 45;
  static const int UbicacioNoMagatzem  = 46;

private:
  #include "terminal.rep"
};
#endif
```

8 Errors

Aquest fitxer conté els missatges d'error usats en la gestió d'errors.

```
10 ubicacio Ubicacio incorrecta.

20 contenidor Matricula incorrecta.
21 contenidor Longitud incorrecta.

30 cataleg Clau es l'string buit.
31 cataleg Clau inexistent.

40 terminal Numero de fileres incorrecte.
41 terminal Numero de places incorrecte.
42 terminal Alcada maxima incorrecta.
43 terminal Estrategia incorrecta.
44 terminal Matricula duplicada.
45 terminal Matricula inexistent.
46 terminal Ubicacio no pertany al magatzem.
```

9 Documentació

Els fitxers lliurats han d'estar degudament documentats. És molt important descriure amb detall i precisió la representació escollida en el fitxer `.rep`, justificant les eleccions fetes, així com les operacions de cada classe. És especialment important explicar amb detall les representacions i els motius de la seva elecció enfront a possibles alternatives, i els algorismes utilitzats.

El cost en temps i en espai és freqüentment el criteri determinant en l'elecció, per la qual cosa s'hauran de detallar aquests costs en la justificació (sempre que això sigui possible) per a cada alternativa considerada i per a l'opció finalment escollida. A més caldrà detallar en el fitxer `.cpp` el cost de cada mètode públic i privat.

En definitiva heu de:

- comentar adequadament el codi, evitant comentaris inútils i superflus
- indicar, en la mesura que sigui possible, el cost dels mètodes de les classes (tant públics com privats)
- descriure amb detall i precisió la representació escollida i justifiqueu l'elecció respecte d'altres.

Un cop enviats els fitxers per via electrònica, aquests seran impresos per a la seva avaluació. No haureu d'imprimir-los vosaltres. No haureu de lliurar cap altra documentació. Per tal d'unificar l'aspecte visual del codi fem servir una eina de *prettyprinting* anomenada `astyle`. Podeu comprovar els resultats que produeix el *prettyprinter* mitjançant la comanda

```
% astyle --style=kr -s2 < fitxer.cpp > fitxer.formatejat
```

i a continuació podeu convertir-lo en un PDF per visualitzar-lo o imprimir-lo

```
% a2ps fitxer.formatejat -o - | ps2pdf - fitxer.pdf
```