```
#include "terminal.hpp"

//-----------------------------------------------------
//
//  MÃ©todes Privats
//
//-----------------------------------------------------

// A partir d'una filera, busca la seguent posiciÃ³ on es poden inserir els segu
ents contenidors de
// 10, 20 i 30 peus respectivament
void terminal::actualitza_pos(int fil)
{
  bool trobat10 = false, trobat20 = false, trobat30 = false;
  int placa = 0, x;

  // Comprovem si es necesari actualitzar les posicions.
  if(_u10.filera() < fil and _u10.filera() != -1) trobat10 = true;
  if((_u20.filera() < fil and _u20.filera() != -1) or (_u20.filera() == -1 and _
m < 2)) trobat20 = true;
  if((_u30.filera() < fil and _u30.filera() != -1) or (_u30.filera() == -1 and _
m < 3)) trobat30 = true;

  while(not trobat30 or not trobat20 or not trobat10) {
    if(_p[fil][placa] < _h) {
      if(not trobat10) {
        _u10 = ubicacio(fil, placa, _p[fil][placa]);
        trobat10 = true;
      }
      if(not trobat20 and placa < _m - 1 and _p[fil][placa] == _p[fil][placa+1])
 { // fixme
        _u20 = ubicacio(fil, placa, _p[fil][placa]);
        trobat20 = true;
      }
      if(not trobat30 and placa < _m - 2 and _p[fil][placa] == _p[fil][placa+1]
and _p[fil][placa] == _p[fil][placa+2]) {
        _u30 = ubicacio(fil, placa, _p[fil][placa]);
        trobat30 = true;
      }
    }

    ++placa;
    if(not trobat10) x = 0;
    else if(not trobat20) x = 1;
    else if(not trobat30) x = 2;
    if(placa >= _m - x ) {
      ++fil;
      placa = 0;
      if(fil >= _n) {
        if(not trobat10) _u10 = ubicacio(-1,0,0);
        if(not trobat20) _u20 = ubicacio(-1,0,0);
        if(not trobat30) _u30 = ubicacio(-1,0,0);
        trobat30 = true;
        trobat20 = true;
        trobat10 = true;
      }
    }
  }
}

void terminal::retira_contenidor_superior(const string &m, bool primer)
{
```



```
  ubicacio u = on(m);
  nat l = _c[m].first/10;

  // Mateixa filera <i, j, k>
  nat i = u.filera();
  nat j = u.placa();
  nat k = u.pis();

  // Retirar els contenidors que hi han a sobre primer
  if (k+1 < _h) {
    for (nat x = 0; x < l; x++) {
      string mat = _t[i][j+x][k+1];
      if (mat != "") retira_contenidor_superior(mat,false);
    }
  }

  // Retirar aquest contenidor
  if (k < _h) {
    for (nat x = 0; x < l; x++) {
      // 1. Eliminar de l'area de emmagatzematge
      _t[i][j+x][k] = "";

      // 2. Actualitzar estructura auxiliar _p
      --_p[u.filera()][u.placa() + x];
    }

    // 3. Actualitzar cataleg
    if (primer) {
      _c.elimina(m);
    } else {
      std::pair<nat, ubicacio> p = std::make_pair(l, ubicacio(-1,0,0));
      _c.assig(m, p);

      // 4. Afegir a l'area d'espera
      contenidor c = contenidor(m,l*10);
      _areaEspera.push_back(c);
      if (c.longitud() == 10) _c10++;
      if (c.longitud() == 20) _c20++;
      if (c.longitud() == 30) _c30++;
    }

    // 5. Indicar nova operacio grua
    _opsGrua++;

    // Nomes pel contenidor base
    if (primer) {
      // 6. Buscar seguent ubicacio lliure
      actualitza_pos(u.filera());

      // 5. Actualizar fragmentacio
      act_fragmentacio(u.filera());

      // 7. Recolocar contenidors del Area d'espera
      recolocarAreaEspera();
    }
  }
}

void terminal::act_fragmentacio(const nat& filera)
{
  _f -= _fFila[filera];
  _fFila[filera] = 0;
```

```
  bool desnivell = true;

  for (nat i = 0; i < _m; i++) {
    if (desnivell) {
      if(i < _m-1) {
        if ( _p[filera][i] != _p[filera][i+1] ) {
          if (_p[filera][i] != _h) ++_fFila[filera];
        } else {
          desnivell = false;
        }
      } else if (i == _m-1 and _p[filera][i] != _h) {
        ++_fFila[filera];
      }
    } else if ( (i < _m-1) and (_p[filera][i] != _p[filera][i+1]) ) {
      desnivell = true;
    }
  }
  _f += _fFila[filera];
}

void terminal::recolocarAreaEspera()
{
  if (_areaEspera.size()) {
    ubicacio areaEspera(-1,0,0);
    list<contenidor>::const_iterator it;
    bool fi = false, b10 = true, b20 = true, b30 = true;
    it = _areaEspera.end();
    --it;

    while(not fi and (b10 or b20 or b30)) {
      if(_c10 == 0 or _u10 == areaEspera) b10 = false;
      if(_c20 == 0 or _u20 == areaEspera) b20 = false;
      if(_c30 == 0 or _u30 == areaEspera) b30 = false;

      if (b10 and (*it).longitud() == 10) {
        _areaEspera.remove(*it);
        --_c10;
        insereix_contenidor(*it);
      } else if (b20 and (*it).longitud() == 20) {
        _areaEspera.remove(*it);
        --_c20;
        insereix_contenidor(*it);
      } else if (b30 and (*it).longitud() == 30) {
        _areaEspera.remove(*it);
        --_c30;
        insereix_contenidor(*it);
      }

      if (it == _areaEspera.begin()) fi = true;
      --it;
    }
  }
}

//---------------------------------------------------
//
//  MÃ©todes de Classe
//
//---------------------------------------------------

terminal::terminal(nat n, nat m, nat h, estrategia st) throw(error) : _c(n*m*h),
 _u10(0,0,0), _u20(0,0,0), _u30(0,0,0)
```



```
{
  if(n == 0) throw error(NumFileresIncorr);
  else _n = n;
  if(m == 0) throw error(NumPlacesIncorr);
  else _m = m;
  if(h == 0 or h > HMAX) throw error(AlcadaMaxIncorr);
  else _h = h;
  if(st != FIRST_FIT and st != LLIURE) throw error(EstrategiaIncorr);
  else _st = st;

  if (m < 3) {
    _u30 = ubicacio(-1,0,0);
    if (m < 2) {
      _u20 = ubicacio(-1,0,0);
    }
  }

  _t = new string**[_n];
  _p = new int*[_n];
  _fFila = new nat[_n];
  for(int i = 0; i < _n; ++i) {
    _t[i] = new string*[_m];
    _p[i] = new int[_m];
    if (_m == 1) _fFila[i] = 1;
    else _fFila[i] = 0;
    for (int j = 0; j < _m; ++j) {
      _t[i][j] = new string[_h];
      _p[i][j] = 0;
      for(int k = 0; k<_h; k++) {
        _t[i][j][k] = "";
      }
    }
  }

  if (_m == 1) _f = _n;
  else _f = 0;
  _opsGrua = 0;
  _c10 = 0;
  _c20 = 0;
  _c30 = 0;
}

/* Constructora per cÃ²pia, assignaciÃ³ i destructora. */
terminal::terminal(const terminal& b) throw(error) : _c(1), _u10(0,0,0), _u20(0,
0,0), _u30(0,0,0)
{
  _n = b._n;
  _m = b._m;
  _h = b._h;
  _st = b._st;
  _t = b._t;
  _p = b._p;
  _u10 = b._u10;
  _u20 = b._u20;
  _u30 = b._u30;
  _areaEspera = b._areaEspera;
  _opsGrua = b._opsGrua;
  _c10 = b._c10;
  _c20 = b._c20;
  _c30 = b._c30;
}
```

```
terminal& terminal::operator=(const terminal& b) throw(error)
{
  _n = b._n;
  _m = b._m;
  _h = b._h;
  _st = b._st;
  _t = b._t;
  _p = b._p;
  _u10 = b._u10;
  _u20 = b._u20;
  _u30 = b._u30;
  _areaEspera = b._areaEspera;
  _opsGrua = b._opsGrua;
  _c10 = b._c10;
  _c20 = b._c20;
  _c30 = b._c30;
  return *this;
}

terminal::~terminal() throw()
{
  for(int i = 0; i < _n; ++i) {
    delete _p[i];
    for (int j = 0; j < _m; ++j) {
      delete[] _t[i][j];
    }
    delete _t[i];
  }
  delete[] _t;
  delete[] _p;
}

/* ColÂ•loca el contenidor c en l'Ã rea d'emmagatzematge de la terminal o
   en l'Ã rea d'espera si no troba lloc en l'Ã rea d'emmagatzematge usant
   l'estratÃ¨gia prefixada en el moment de crear la terminal. Si el
   contenidor c es colÂ•loca en l'Ã rea d'emmagatzematge pot succeir que
   un o mÃ©s contenidors de l'Ã rea d'espera puguin ser moguts a l'Ã rea
   d'emmagatzematge.
   En aquest cas es mouran els contenidors de l'Ã rea d'espera a l'Ã rea
   d'emmagatzematge seguint l'ordre que indiqui l'estratÃ¨gia que s'estÃ
   usant. Finalment, genera un error si ja existÃ-s a la terminal un
   contenidor amb una matrÃ-cula idÃ¨ntica que la del contenidor c. */
void terminal::insereix_contenidor(const contenidor &c) throw(error)
{
  ubicacio u = on(c.matricula());
  if(u == ubicacio(-1,-1,-1) or u == ubicacio(-1,0,0)) {
    u = ubicacio(-1,0,0);
    if(_st == FIRST_FIT) {
      if(c.longitud() == 10) {
        if(_u10 != u) {
          _t[_u10.filera()][_u10.placa()][_u10.pis()] = c.matricula();
          ++_p[_u10.filera()][_u10.placa()];
          u = _u10;
          ++_opsGrua;
          act_fragmentacio(_u10.filera());
        } else {
          _areaEspera.push_back(c);
          ++_c10;
        }
      } else if(c.longitud() == 20) {
        if(_u20 != u) {
          _t[_u20.filera()][_u20.placa()][_u20.pis()] = c.matricula();
          _t[_u20.filera()][_u20.placa()+1][_u20.pis()] = c.matricula();
          ++_p[_u20.filera()][_u20.placa()];
          ++_p[_u20.filera()][_u20.placa()+1];
          u = _u20;
          ++_opsGrua;
          act_fragmentacio(_u20.filera());
        } else {
          _areaEspera.push_back(c);
          ++_c20;
        }
      } else {
        if(_u30 != u) {
          _t[_u30.filera()][_u30.placa()][_u30.pis()] = c.matricula();
          _t[_u30.filera()][_u30.placa()+1][_u30.pis()] = c.matricula();
          _t[_u30.filera()][_u30.placa()+2][_u30.pis()] = c.matricula();
          ++_p[_u30.filera()][_u30.placa()];
          ++_p[_u30.filera()][_u30.placa()+1];
          ++_p[_u30.filera()][_u30.placa()+2];
          u = _u30;
          ++_opsGrua;
          act_fragmentacio(_u30.filera());
        } else {
          _areaEspera.push_back(c);
          ++_c30;
        }
      }
      std::pair<nat, ubicacio> p = std::make_pair(c.longitud(), u);
      _c.assig(c.matricula(), p);
      if (_u10.filera() != -1)
        actualitza_pos(_u10.filera());
      recolocarAreaEspera();
    } else { // Altra estrategia

    }
  } else throw error(MatriculaDuplicada);
}
/* Retira de la terminal el contenidor c la matrÃ-cula del qual Ã©s igual
   a m. Aquest contenidor pot estar a l'Ã rea d'emmagatzematge o a l'Ã rea
   d'espera. Si el contenidor estiguÃ©s a l'Ã rea d'emmagatzematge llavors
   s'hauran de moure a l'Ã rea d'espera tots els contenidors que siguin
   necessaris per netejar la part superior de c, s'hauran de retirar
   possiblement diversos contenidors, comenÃ§ant pel contenidor sense cap
   altre a sobre amb el nÃºmero de plaÃ§a mÃ©s baix (mÃ©s a l'esquerra) i aixÃ-
   successivament (veure exemple amb detall a la subseccioÃ³ EstratÃ¨gia
   FIRST_FIT). Un cop s'hagi eliminat el contenidor indicat, s'intenta
   moure contenidors de l'Ã rea d'espera a l'Ã rea d'emmagatzematge, seguint
   l'ordre que indiqui l'estratÃ¨gia que s'estÃ  usant. Genera un error si a
   la terminal no hi ha cap contenidor la matrÃ-cula del qual sigui igual a m. *
/
void terminal::retira_contenidor(const string &m) throw(error)
{
  ubicacio areaEspera(-1,0,0);
  ubicacio u = on(m);

  if (u != areaEspera) {
    if (_c.existeix(m)) {
      retira_contenidor_superior(m,true);
    } else {
      throw error(MatriculaInexistent);
    }
  } else {
    list<contenidor>::const_iterator it;
```

```
    bool fi = false;
    it = _areaEspera.begin();
    while(it != _areaEspera.end() and not fi) {
      if((*it).matricula() == m) {
        _areaEspera.remove(*it);
        _c.elimina(m);
        fi = true;
      } else {
        ++it;
      }
    }
  }
}

/* Retorna la ubicaciÃ³ <i, j, k> del contenidor la matrÃ-cula del qual Ã©s
   igual a m si el contenidor estÃ  a l'Ã rea d'emmagatzematge de la terminal,
   la ubicaciÃ³ <-1, 0, 0> si el contenidor estÃ  a l'Ã rea d'espera, i la
   ubicaciÃ³ <-1, -1, -1> si no existeix cap contenidor que tingui una
   matrÃ-cula igual a m.
   Cal recordar que si un contenidor tÃ© mÃ©s de 10 peus, la seva ubicaciÃ³
   correspon a la plaÃ§a que tingui el nÃºmero de plaÃ§a mÃ©s petit. */
ubicacio terminal::on(const string &m) const throw()
{
  try {
    return _c[m].second;
  } catch (...) {
    return ubicacio(-1,-1,-1);
  }
}

/* Retorna la longitud del contenidor la matrÃ-cula del qual Ã©s igual
   a m. Genera un error si no existeix un contenidor a la terminal
   la matrÃ-cula del qual sigui igual a m. */
nat terminal::longitud(const string &m) const throw(error)
{
  try {
    return _c[m].first;
  } catch (...) {
    throw error(MatriculaInexistent);
  }
}

/* Retorna la matrÃ-cula del contenidor que ocupa la ubicaciÃ³ u = <i, j, k>
   o la cadena buida si la ubicaciÃ³ estÃ  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 ubicaciÃ³ vÃ lida de l'Ã rea d'emmagatzematge. Cal observar
   que si m, obtinguda amb t.contenidor_ocupa(u, m), Ã©s una matrÃ-cula (no
   la cadena buida) pot succeir que u != t.on(m), ja que un contenidor pot
   ocupar diverses places i la seva ubicaciÃ³ es correspon amb la de la
   plaÃ§a ocupada amb nÃºmero de plaÃ§a mÃ©s baix. */
void terminal::contenidor_ocupa(const ubicacio &u, string &m) const throw(error)
{
  int i = u.filera();
  int j = u.placa();
  int k = u.pis();

  try {
    if ( (i < _n) and (j < _m) and (k < _h) ) {
      m = _t[i][j][k];
    } else {
      throw error();
    }
```



```
  } catch (...) {
    throw error(UbicacioNoMagatzem);
  }
}

/* Retorna el nombre de places de la terminal que en aquest instant
   nomÃ©s hi cabrien un contenidor de 10 peus, perÃ² no un de mÃ©s llarg.
   Per exemple, la filera de la figura 1 de l'enunciat contribuirÃ  amb
   7 unitats a la fragmentaciÃ³ 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 terminal::fragmentacio() const throw()
{
  return _f;
}

/* Retorna el nÃºmero d'operacions de grua realitzades des del moment
   de creaciÃ³ de la terminal.
   Es requereix d'una operaciÃ³ de grua per moure un contenidor
   des de l'Ã rea d'espera a l'Ã rea d'emmagatzematge o viceversa.
   TambÃ© es requereix d'una operaciÃ³ de grua per inserir o
   retirar directament un contenidor de l'Ã rea d'emmagatzematge.
   En canvi no requereix cap operaciÃ³ de grua inserir o
   retirar directament un contenidor de l'Ã rea d'espera. */
nat terminal::ops_grua() const throw()
{
  return _opsGrua;
}

/* Retorna la llista de les matrÃ-cules de tots els contenidors
   de l'Ã rea d'espera de la terminal, en ordre alfabÃ¨tic creixent. */
void terminal::area_espera(list<string> &l) const throw()
{
  list<contenidor>::const_iterator it;
  for (it = _areaEspera.begin(); it != _areaEspera.end(); ++it) {
    l.push_back((*it).matricula());
  }

  l.sort();
}

/* Retorna el nÃºmero de fileres de la terminal. */
nat terminal::num_fileres() const throw()
{
  return _n;
}

/* Retorna el nÃºmero de places per filera de la terminal. */
nat terminal::num_places() const throw()
{
  return _m;
}

/* Retorna l'alÃ§ada mÃ xima d'apilament de la terminal. */
nat terminal::num_pisos() const throw()
{
  return _h;
}

/* Retorna l'estratÃ¨gia d'inserciÃ³ i retirada de contenidors de la terminal. *
/
terminal::estrategia terminal::quina_estrategia() const throw()
```

```
{
  return _st;
}
```