```
#include "call_registry.hpp"

/* Construeix un call_registry buit. */
call_registry::call_registry() throw(error)
{
  _M = 4;
  _quants = 0;
  _taula = new node_hash *[_M];
  for(int i = 0; i<_M; ++i) {
    _taula[i] = nullptr;
  }
}

/* Constructor per cÃ²pia, operador d'assignaciÃ³ i destructor. */
call_registry::call_registry(const call_registry& R) throw(error)
{
  _M = R._M;
  _quants = R._quants;
  _taula = new node_hash *[_M];
  for(int i = 0; i<_M; ++i) {
    node_hash *n = R._taula[i];
    node_hash *ant = nullptr;
    _taula[i] = nullptr;
    while(n!=nullptr) {
      if (ant!=nullptr) {
        node_hash *nou = new node_hash;
        nou->_p = n->_p;
        nou->_seg = nullptr;
        ant->_seg = nou;
        ant = nou;

      } else {
        node_hash *nou = new node_hash;
        nou->_p = n->_p;
        nou->_seg = nullptr;
        ant = nou;
        _taula[i] = ant;
      }
      n = n->_seg;
    }

  }
}

call_registry& call_registry::operator=(const call_registry& R) throw(error)
{
  this->~call_registry();
  _M = R._M;
  _quants = R._quants;
  _taula = new node_hash *[_M];
  for(int i = 0; i<_M; ++i) {
    node_hash *n = R._taula[i];
    node_hash *ant = nullptr;
    _taula[i] = nullptr;
    while(n!=nullptr) {
      if (ant!=nullptr) {
        node_hash *nou = new node_hash;
        nou->_p = n->_p;
        nou->_seg = nullptr;
        ant->_seg = nou;
        ant = nou;
```



```
      } else {
        node_hash *nou = new node_hash;
        nou->_p = n->_p;
        nou->_seg = nullptr;
        ant = nou;
        _taula[i] = ant;
      }
      n = n->_seg;
    }

  }
  return *this;
}

call_registry::~call_registry() throw()
{
  for(int i = 0; i < _M; ++i) {
    node_hash *current = _taula[i];
    while (current != nullptr) {
      node_hash *temp = current;
      current = current->_seg;
      delete temp;
    }
  }
  delete[] _taula;
}

/* Registra que s'ha realitzat una trucada al nÃºmero donat,
incrementant en 1 el comptador de trucades associat. Si el nÃºmero no
estava prÃ¨viament en el call_registry afegeix una nova entrada amb
el nÃºmero de telÃ¨fon donat, l'string buit com a nom i el comptador a 1. */
void call_registry::registra_trucada(nat num) throw(error)
{
  int pos = h(num) % _M;
  if (_taula[pos] == nullptr) {
    node_hash *element = new node_hash;
    phone telefon(num,"", 1);
    element->_p = telefon;
    element->_seg = nullptr;
    _taula[pos] = element;
    _quants++;
    float fc = factor_de_carrega();
    if(fc > 0.8) redispersiÃ³(fc);
  } else {
    bool trobat = false;
    node_hash *element = _taula[pos];
    node_hash *ant = nullptr;
    while(element != nullptr and not trobat and element->_p.numero()<=num) {
      if(element->_p.numero()==num) trobat = true;
      else {
        ant = element;
        element=element->_seg;
      }
    }
    if(not trobat) {
      node_hash *nou = new node_hash;
      phone telefon(num,"", 1);
      nou->_p = telefon;
      nou->_seg = element;
      if(ant == nullptr) _taula[pos] = nou;
      else ant->_seg = nou;
```

```
      _quants++;
      float fc = factor_de_carrega();
      if(fc > 0.8) redispersiÃ³(fc);
    } else { //Si hi ha el telefon al call registry freq++
      ++(element->_p);
    }
  }
}

/* Assigna el nom indicat al nÃºmero donat.
Si el nÃºmero no estava prÃ¨viament en el call_registry, s'afegeix
una nova entrada amb el nÃºmero i nom donats, i el comptador
de trucades a 0.
Si el nÃºmero existia prÃ¨viament, se li assigna el nom donat. */
void call_registry::assigna_nom(nat num, const string& name) throw(error)
{
  int pos = (h(num))%_M;
  bool trobat  = false;
  node_hash * element = _taula[pos];
  node_hash * ant = nullptr;
  while(element != nullptr and not trobat and element->_p.numero() <= num) {
    if(element->_p.numero() == num) trobat = true;
    else {
      ant = element;
      element = element->_seg;
    }
  }
  if(trobat) {
    int f = element->_p.frequencia();
    phone nou(num,name,f);
    element->_p = nou;
  } else {
    node_hash *nou = new node_hash;
    phone telefon(num,name, 0);
    nou->_p = telefon;
    nou->_seg = element;
    if(ant == nullptr) _taula[pos] = nou;
    else ant->_seg = nou;
    _quants++;
    float fc = factor_de_carrega();
    if(fc > 0.8) redispersiÃ³(fc);
  }
}

/* Elimina l'entrada corresponent al telÃ¨fon el nÃºmero de la qual es dÃ³na.
Es produeix un error si el nÃºmero no estava present. */
void call_registry::elimina(nat num) throw(error)
{
  int pos = (h(num))%_M;
  bool trobat = false;
  node_hash *element = _taula[pos];
  node_hash *ant = nullptr;
  while(element != nullptr and not trobat and element->_p.numero()<=num) {
    if(element->_p.numero()==num) trobat = true;
    else {
      ant = element;
      element=element->_seg;
    }
  }
  if (trobat) {
    if(ant==nullptr) {
      _taula[pos] = element->_seg;
```



```
      delete element;
      --_quants;
      float fc = factor_de_carrega();
      //cout << fc << endl;
      if(fc < 0.3) redispersiÃ³(fc);
    } else {
      ant->_seg = element->_seg;
      delete element;
      --_quants;
      float fc = factor_de_carrega();
      //cout << fc << endl;
      if(fc < 0.3) redispersiÃ³(fc);
    }
  } else throw error(ErrNumeroInexistent);
}

/* Retorna cert si i nomÃ©s si el call_registry contÃ© un
telÃ¨fon amb el nÃºmero donat. */
bool call_registry::conte(nat num) const throw()
{
  int pos = (h(num))%_M;
  bool trobat  =false;
  node_hash * element = _taula[pos];
  while(element!=NULL and not trobat and element->_p.numero()<=num) {
    if(element->_p.numero()==num) trobat = true;
    else {
      element=element->_seg;
    }
  }
  return trobat;

}

/* Retorna el nom associat al nÃºmero de telÃ¨fon que s'indica.
Aquest nom pot ser l'string buit si el nÃºmero de telÃ¨fon no
tÃ© un nom associat. Es produeix un error si el nÃºmero no estÃ  en
el call_registry. */
string call_registry::nom(nat num) const throw(error)
{
  int pos = (h(num))%_M;
  bool trobat  =false;
  node_hash * element = _taula[pos];
  while(element!=NULL and not trobat and element->_p.numero()<=num) {
    if(element->_p.numero()==num) trobat = true;
    else {
      element=element->_seg;
    }
  }
  if (not trobat)
    throw error(ErrNumeroInexistent);
  else
    return element->_p.nom();
}

/* Retorna el comptador de trucades associat al nÃºmero de telÃ¨fon
indicat. Aquest nÃºmero pot ser 0 si no s'ha efectuat cap trucada a
aquest nÃºmero. Es produeix un error si el nÃºmero no estÃ  en el
call_registry. */
nat call_registry::num_trucades(nat num) const throw(error)
{
  int pos = (h(num))%_M;
  bool trobat  =false;
```

```
  node_hash * element = _taula[pos];
  while(element!=nullptr and not trobat and element->_p.numero()<=num) {
    if(element->_p.numero()==num) trobat = true;
    else {
      element=element->_seg;
    }
  }
  if (not trobat)
    throw error(ErrNumeroInexistent);
  else
    return element->_p.frequencia();
}

/* Retorna cert si i nomÃ©s si el call_registry estÃ  buit. */
bool call_registry::es_buit() const throw()
{
  return _quants == 0;
}

/* Retorna quants nÃºmeros de telÃ¨fon hi ha en el call_registry. */
nat call_registry::num_entrades() const throw()
{
  return _quants;
}

/* Fa un bolcat de totes les entrades que tenen associat un
nom no nul sobre un vector de phone.
Comprova que tots els noms dels telÃ¨fons siguin diferents;
es produeix un error en cas contrari. */
void call_registry::dump(vector<phone>& V) const throw(error)
{
  nat j=0;
  vector<string> noms;
  for(int i = 0; i<_M; ++i) {
    node_hash * element = _taula[i];
    while(element!=NULL) {
      if(element->_p.nom()!="") {
        noms.push_back(element->_p.nom());
        V.push_back(element->_p);
      }
      element = element->_seg;

    }
  }
  ordena(noms);
  bool repetits = false;
  if (noms.size()>1) {
    while(not repetits and j<noms.size()-1) {
      if(noms[j]==noms[j+1]) repetits =true;
      j++;
    }
    if (repetits) {
      throw error(ErrNomRepetit);
      for(unsigned int i =0; i<V.size(); i++) {
        V.pop_back();
      }
    }
  }
}

// MÃ¨todes privats
long call_registry::h(int k)
```



```
{
  long i = ((k * k * MULT) << 20) >> 4;
  if (i < 0)
    i = -i;
  return i;
};

float call_registry::factor_de_carrega() const
{
  float fc = ((float)this->_quants/(float)this->_M);
  return fc;
};

void call_registry::esborra_taula(node_hash **t, nat mida)
{

  for (nat i = 0; i < mida; ++i) {
    node_hash *current = t[i];
    while (current != nullptr) {
      node_hash *temp = current;
      current = current->_seg;
      delete temp;
    }
  }

  delete[] t;
};

void call_registry::redispersiÃ³(float fc)
{
  //cout << _M << endl;
  if(fc > 0.8) {
    nat m_aux = 2*(this->_M)+1;
    node_hash ** t_aux = new node_hash *[m_aux];
    for(int i = 0; i<m_aux; ++i) {
      t_aux[i] = nullptr;
    }
    swap(_taula, t_aux);
    swap(_M, m_aux);
    _quants = 0;
    for(int i=0; i<m_aux; ++i) {
      node_hash *n = t_aux[i];
      while(n != nullptr) {
        afegeix_numero(n->_p);
        n=n->_seg;
       }
    }
    esborra_taula(t_aux, m_aux);
  } // ((this->_M+1)/2);
  else if(fc < 0.3) {
    nat m_aux = (this->_M+1)/2;
    node_hash ** t_aux = new node_hash *[m_aux];
    for(int i = 0; i<m_aux; ++i) {
      t_aux[i] = nullptr;
    }
    swap(_taula, t_aux);
    swap(_M, m_aux);
    _quants = 0;
    for(int i=0; i<m_aux; ++i) {
      node_hash *n = t_aux[i];
      while(n != nullptr) {
        afegeix_numero(n->_p);
```

```
        n=n->_seg;
      }
    }
    esborra_taula(t_aux, m_aux);
  }


};

void call_registry::ordena(vector<string>& V) const
{
  if(V.size()<2) return;
  vector<string> a = V;
  vector<string> b;
  parteix(a,b);
  ordena(a);
  ordena(b);
  V = fusiona(a,b);
};

void call_registry::parteix(vector<string>& a, vector<string>& b) const
{
  //cout << "parteix" <<endl;
  int  mida = int(a.size()/2);
  int n = int(a.size()) -1;
  for(unsigned int i = n; i >= mida; i--) {
    b.push_back(a[i]);
    a.pop_back();
  }
  //cout << size(b) << endl;
}

vector<string> call_registry::fusiona(const vector<string>& a, const vector<stri
ng>& b) const
{
  //cout << "fusiona" <<endl;
  vector<string> res;
  int sa = int(a.size());
  int sb = int(b.size());
  //cout << sb << endl;
  int ia = 0;
  int ib = 0;
  while((ia<sa) and (ib<sb)) {
    //cout << "m" << endl;
    if(a[ia]<b[ib]) {
      //cout << a[ia] << endl;
      res.push_back(a[ia]);
      ia++;
    } else {
      res.push_back(b[ib]);
      ib++;
    }
  }
  while(ia<sa) {
    //cout << "hola";
    res.push_back(a[ia]);
    ia++;
  }

  while(ib<sb) {
    res.push_back(b[ib]);
    ib++;
```



```
  }
  return res;
}

void call_registry::afegeix_numero(phone p)
{
  int pos = h(p.numero()) % _M;
  if (_taula[pos] == nullptr) {
    node_hash *element = new node_hash;
    element->_p = p;
    element->_seg = nullptr;
    _taula[pos] = element;
    _quants++;
  } else {
    bool trobat = false;
    node_hash *element = _taula[pos];
    node_hash *ant = nullptr;
    while(element != nullptr and not trobat and element->_p.numero()<=p.numero()
) {
      if(element->_p.numero()==p.numero()) trobat = true;
      else {
        ant = element;
        element=element->_seg;
      }
    }
    if(not trobat) {
      node_hash *nou = new node_hash;
      nou->_p = p;
      nou->_seg = element;
      if(ant == nullptr) _taula[pos] = nou;
      else ant->_seg = nou;
      _quants++;
    } else { //Si hi ha el telefon al call registry freq++
      ++(element->_p);
    }
  }
}
```