```
#include "phone.hpp"

// RepresentaciÃ³ de Theta amb 0

phone::phone(nat num, const string& name, nat compt) throw(error)
{
  // Cost: 0(n), on n es la longitud de l'string 'name'
  for(nat i=0; i<name.size(); i++) {
    if(name[i]==DELETECHAR or name[i]==ENDCHAR or name[i]==ENDPREF)
      throw error(ErrNomIncorrecte);
  }
  name_phone = name;
  freq= compt;
  number = num;
}

/* Tres grans. Constructor per copia, operador d'assignacio i destructor. */
phone::phone(const phone& T) throw(error)
{
  // Cost: 0(1). Constant
  name_phone = T.name_phone;
  number = T.number;
  freq = T.freq;
}

phone& phone::operator=(const phone& T) throw(error)
{
  // Cost: 0(1). Constant
  name_phone = T.name_phone;
  number = T.number;
  freq = T.freq;
  return *this;
}

phone::~phone() throw()
{
  // Cost: 0(1). Constant
}

/* Retorna el numero de telefon. */
nat phone::numero() const throw()
{
  // Cost 0(1). Constant
  return number;
}

/* Retorna el name associat al telefon, eventualment l'string buit. */
string phone::nom() const throw()
{
  // Cost 0(1). Constant
  return name_phone;
}

/* Retorna el numero de vegades que s'ha trucat al telefon. */
nat phone::frequencia() const throw()
{
  // Cost 0(1). Constant
  return freq;
}

/* Operador de preincrement. */
phone& phone::operator++() throw()
```



```
{
  // Cost 0(1). Constant
  freq = freq + 1;
  return *this;
}

/* Operador de postincrement. */
phone phone::operator++(int) throw()
{
  // Cost 0(1). Constant
  phone aux = *this;
  freq = freq + 1;
  return aux;
}

/* Operadors de comparacio.  */

bool phone::operator==(const phone& T) const throw()
{
  // Cost 0(1). Constant
  bool igual = false;
  if(freq == T.freq and name_phone==T.name_phone) igual = true;
  return igual;
}

bool phone::operator!=(const phone& T) const throw()
{
  // Cost 0(1). Constant
  return not (*this==T);
}

bool phone::operator<(const phone& T) const throw()
{
  // Cost 0(1). Constant
  bool petit = false;
  if (freq<T.freq) petit = true;
  else if(freq == T.freq and name_phone<T.name_phone) petit = true;
  return petit;
}

bool phone::operator>(const phone& T) const throw()
{
  // Cost 0(1). Constant
  bool gran = false;
  if (freq>T.freq) gran = true;
  else if(freq == T.freq and name_phone>T.name_phone) gran = true;
  return gran;
}

bool phone::operator<=(const phone& T) const throw()
{
  // Cost 0(1). Constant
  return (*this<T) or  (*this==T);
}
bool phone::operator>=(const phone& T) const throw()
{
  // Cost 0(1). Constant
  return (*this>T) or (*this==T);
}

/* Caracters especials no permesos en un name de telefon. */
static const char DELETECHAR = '<';
```

```
static const char ENDCHAR = '|';
static const char ENDPREF = '\0';

/* Gestio d'errors. */
static const int  ErrNomIncorrecte = 11;


// g++ phone.cpp -o phone.e -O0 -g -Wall -Wextra -Werror -Wno-sign-compare -std=
c++14 -ansi -I /home/bruna/ESIN/projecte/incl -lesin
```