Projecte de ESIN
Normativa i Enunciat

Tardor de 2019

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 bcasas@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++-5.4.0`) 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 `ahto.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 9 de gener de 2020 a les 23:55. 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

Un agenda telefònica és una aplicació comuna en telèfons mòbils, smarthphones i ordinadors. En general una persona gestiona un agenda amb una quantitat reduïda de números de telèfon, però tot i això aquest tipus d'aplicacions ofereixen poca "intel·ligència". En els telèfons mòbils és habitual que l'única facilitat de cerca consisteixi en prémer la lletra inicial del nom associat al número de telèfon i, un cop situats sobre el primer dels noms que comencen amb la lletra en qüestió, hom s'hagi de desplaçar seqüencialment per la llista fins trobar el contacte que li interessa.

En aquest projecte desenvoluparem el sistema `EasyDial` que, si bé té algunes limitacions, exhibeix una major "intel·ligència" i proporciona major comoditat a l'usuari. Amb alguns esforços addicionals i una interfície d'usuari agradable podria donar origen a una aplicació veritablement pràctica.

Una part del sistema permetria registrar números de telèfon, assignar noms a aquests números de telèfon i comptabilitzar quantes trucades s'efectuen a cadascun dels telèfons.

L'altra part del sistema, la realment "innovadora", construeix una estructura de dades a partir de la informació recol·lectada per l'altre subsistema. A partir d'aquí ens permet consultar números de telèfon introduint un prefix mínim del nom. El sistema tindrà en compte la freqüència amb que acostumem a trucar a cada número de telèfon per fer la cerca el més ràpida i còmoda possible.

Un exemple concret ens pot ajudar a entendre com funciona `EasyDial`. Per fixar idees, suposem que l'aplicació està instal·lada en un mòbil i que la guia conté informació dels telèfons que es poden veure a la figura 1.

```
MARIA  972261435   15 trucades
JOSEP  934578916   50 trucades
MAR    934907288    5 trucades
MIQUEL 666931459   30 trucades
MARTA  678818034   10 trucades
```

Figura 1: Exemple d'agenda de telèfons

Premem el botó de trucada. Donat que JOSEP és a qui més truquem, el sistema ens presenta en pantalla aquest nom. Si realment volguéssim trucar-lo només hauríem de tornar a prémer el botó de trucada (o un botó de OK) i llavors realitzaria efectivament la trucada. Però suposem que no, que volem trucar a MARTA. Així que premem la 'M', i llavors `EasyDial` ens presenta MIQUEL, ja que és el telèfon usat més freqüentment que comença per aquesta lletra. Novament podríem donar-li a OK si aquest fos el telèfon al que volíem trucar, però com no és així, premem 'A'. `EasyDial` escriu llavors MARIA, ja que és el telèfon que comença per 'MA' més freqüent. Però nosaltres premerem 'R'

perquè no volem trucar a MARIA, però sí a un telèfon que comença per 'MAR'. I ara `EasyDial` ens proposaria MARTA, perquè és el telèfon més freqüent que comença per 'MAR' i no ha estat rebutjat: en efecte, MARIA també comença per 'MAR' i és més freqüent que MARTA, però si haguéssim volgut trucar a MARIA ja no hauríem premut la 'R' i haguéssim confirmat la trucada a MARIA quan ens va ser proposat.

En total, ha estat suficient prémer tres lletres ('M', 'A', 'R') per localitzar el telèfon de MARTA. Per trucar a MIQUEL només necessitaríem prémer una lletra. En general quant més freqüent sigui un telèfon més senzill serà trobar-lo a l'agenda. Només en el pitjor dels casos hauríem d'introduir el nom complet (o un prefix que el distingeix de tota la resta de noms) per localitzar un telèfon.

És fàcil donar-se compte que el sistema ens donarà molt bones prestacions encara que la quantitat de números de telèfon que hagi emmagatzemats sigui molt elevada.

Resumint, el projecte comptarà, entre d'altres, amb els següents tres mòduls fonamentals:

1. Un mòdul permetrà emmagatzemar i gestionar informació sobre números de telèfon, noms associats i freqüències de trucada a cadascun d'ells.
2. Un altre mòdul contindrà l'algorisme d' "interacció" amb l'usuari que proposarà successius telèfons en funció de l'entrada que proporciona l'usuari.
3. El tercer mòdul extreu i organitza la informació proporcionada pel primer per fer de manera molt eficient la tasca de consulta (guiada pel segon mòdul). De fet, la "intel·ligència" del sistema resideix en aquest mòdul.

En les següents seccions es presenten el disseny modular que s'ha escollit per resoldre el problema i també la definició i descripció de les classes i mòduls que heu d'implementar.

3 Disseny modular

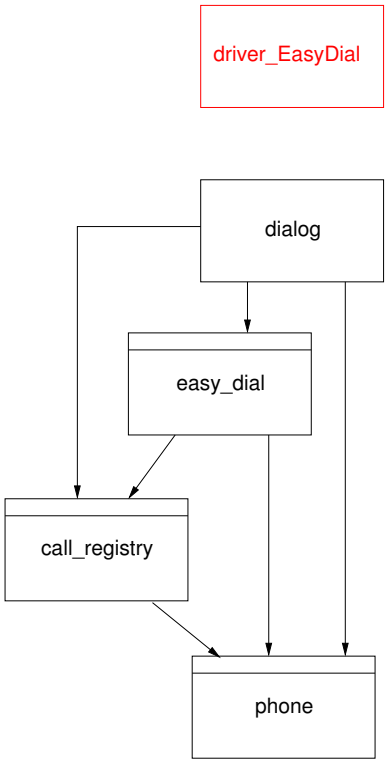

Figura 2: Disseny modular del projecte.

El sistema `EasyDial` consta d'un mòdul funcional anomenat `dialog` en el que es defineix una funció del mateix nom que simula la interacció en la que es tecleja un nom lletra a lletra en un mòbil i la presentació en pantalla dels noms i el número de telèfon final. Per tal de realitzar la seva feina, aquest mòdul usa la classe `easy_dial`, la qual recull la informació acumulada en la classe `call_registry` i l'emmagatzema en una estructura de manera que les consultes siguin eficients. La classe `phone` permet gestionar les dades associades a un telèfon: número, nom i comptador de trucades.

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 iostream 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_EasyDial`. Aquest mòdul conté el programa principal i usa a totes les classes que heu d'implementar, és a dir, permet invocar cada una de les operacions dels diferents mòduls i classes,

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 per 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 una 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 phone.
- Implementar la classe call_registry.
- Implementar la classe easy_dial.
- Implementar la classe dialog.

4 La classe phone

La classe `phone` conté les dades relatives a un telèfon: el número del telèfon, un nom (string) associat al número, i un comptador amb el nombre de vegades que s'ha trucat al telèfon. A aquest comptador se l'anomena indistintament comptador de trucades o comptador de freqüència. La classe també inclou un operador de comparació $>$.

Decisions sobre les dades: Un número de telèfon és un natural. El nom associat a un número és un string format per una seqüència de caràcters del codi ASCII, exceptuant els caràcters `DELETECHAR` = '<', `ENDCHAR` = '|' i `ENDPREF` = '\0'. Els strings no tenen una longitud màxima coneguda, encara que a la majoria dels casos seran "curts". Un string buit és admissible i indica que el número no té (encara) cap nom associat. El comptador associat a un número de telèfon és un natural qualsevol.

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

```
#ifndef _PHONE_HPP
#define _PHONE_HPP

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

using namespace std;
using util::nat;

class phone{
public:

  /* Construeix un telèfon a partir del seu número (num), el seu nom
  (name) i el seu comptador de trucades (compt).
  Es produeix un error si name no és un identificador legal. */
  phone(nat num=0, const string& name="", nat compt=0) throw(error);

  /* Tres grans. Constructor per còpia, operador d'assignació
  i destructor. */
  phone(const phone& t) throw(error);
  phone& operator=(const phone& t) throw(error);
  ~phone() throw();

  /* Retorna el número de telèfon. */
  nat numero() const throw();

  /* Retorna el nom associat al telèfon, eventualment l'string buit. */
  string nom() const throw();
```

```
  /* Retorna el número de vegades que s'ha trucat al telèfon. */
  nat frequencia() const throw();

  /* Operador de preincrement.
  Incrementa en 1 el número de vegades que s'ha trucat al telèfon i
  retorna una referència a aquest telèfon. */
  phone& operator++() throw();

  /* Operador de postincrement.
  Incrementa en 1 el número de vegades que s'ha trucat al telèfon i
  retorna una còpia d'aquest telèfon sense incrementar. */
  phone operator++(int) throw();

  /* Operadors de comparació.  L'operador > retorna cert, si i només si,
  el telèfon sobre el que s'aplica el mètode és més freqüent que el
  telèfon T, o a igual freqüència, el nom associat al telèfon és
  major en ordre lexicogràfic que el nom associat a T.
  La resta d'operadors es defineixen consistentment respecte a >. */
  bool operator==(const phone& t) const throw();
  bool operator!=(const phone& t) const throw();
  bool operator<(const phone& t) const throw();
  bool operator>(const phone& t) const throw();
  bool operator<=(const phone& t) const throw();
  bool operator>=(const phone& t) const throw();

  /* Caràcters especials no permesos en un nom de telèfon. */
  static const char DELETECHAR = '<';
  static const char ENDCHAR = '|';
  static const char ENDPREF = '\0';

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

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

5 La classe call_registry

La classe `call_registry` conté les dades relatives a un conjunt de telèfons. Recordem que un telèfon consta de número, nom associat (que eventualment pot ser l'string buit) i un comptador de trucades. La classe ofereix operacions per assignar un nom a un telèfon, registrar que s'ha produït una trucada, consultes, etc.

L'operació `assigna_nom` pot ser utilitzada per assignar un nom a un telèfon que no tenia un nom associat, modificar el nom associat a un telèfon donat i eliminar el nom associat a un telèfon (assignant l'string buit).

És important destacar que no és un problema que en un moment donat dos o més números tinguin el mateix nom i per això `assigna_nom` no fa comprovacions al respecte.

La classe també inclou una operació de bolcat (`dump`), la qual copia totes les entrades que tenen un nom associat no buit, en un ordre qualsevol, en un `vector` de `phone`s.

Decisions sobre les dades: Un objecte de la classe `call_registry` conté un conjunt de telèfons de mida no acotada, amb la limitació que tots els números de telèfon han de ser diferents.

Implementació: La representació d'aquesta classe es trobarà en el fitxer `call_registry.rep` i la implementació en el fitxer `call_registry.cpp`. Només es pot usar la classe `vector` de la biblioteca estàndard de C++ en el mètode `dump`.

Les operacions d'aquesta classe han de ser totes eficients en el cas pitjor o mitjà. En particular, el constructor per còpia, l'assignació i el destructor han de tenir cost lineal. Tota la resta d'operacions, excepte `dump`, han de tenir cost logarítmic o inferior.

```
#ifndef _CALL_REGISTRY_HPP
#define _CALL_REGISTRY_HPP

#include "phone.hpp"
#include <esin/error>
#include <esin/util>
#include <string>
#include <vector>

using namespace std;
using util::nat;

class call_registry {
public:

  /* Construeix un call_registry buit. */
  call_registry() throw(error);
```

```
/* Constructor per còpia, operador d'assignació i destructor. */
call_registry(const call_registry& cr) throw(error);
call_registry& operator=(const call_registry& cr) throw(error);
~call_registry() throw();

/* Registra que s'ha realitzat una trucada al número donat,
incrementant en 1 el comptador de trucades associat. Si el número
no estava prèviament en el call_registry afegeix una nova entrada
amb el número de telèfon donat, l'string buit com a nom i el
comptador a 1. */
void registra_trucada(nat num) throw(error);

/* Assigna el nom indicat al número donat.
Si el número no estava prèviament en el call_registry, s'afegeix
una nova entrada amb el número i nom donats, i el comptador
de trucades a 0.
Si el número existia prèviament, se li assigna el nom donat. */
void assigna_nom(nat num, const string& name) throw(error);

/* Elimina l'entrada corresponent al telèfon el número de la qual
es dóna. Es produeix un error si el número no estava present. */
void elimina(nat num) throw(error);

/* Retorna cert si i només si el call_registry conté un
telèfon amb el número donat. */
bool conte(nat num) const throw();

/* Retorna el nom associat al número de telèfon que s'indica.
Aquest nom pot ser l'string buit si el número de telèfon no
té un nom associat. Es produeix un error si el número no està en
el call_registry. */
string nom(nat num) const throw(error);

/* Retorna el comptador de trucades associat al número de telèfon
indicat. Aquest número pot ser 0 si no s'ha efectuat cap trucada a
aquest número. Es produeix un error si el número no està en el
call_registry. */
nat num_trucades(nat num) const throw(error);

/* Retorna cert si i només si el call_registry està buit. */
bool es_buit() const throw();

/* Retorna quants números de telèfon hi ha en el call_registry. */
nat num_entrades() const throw();

/* 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 telèfons siguin diferents;
```

```
  es produeix un error en cas contrari. */
  void dump(vector<phone>& V) const throw(error);

  /* Gestió d'errors. */
  static const int  ErrNumeroInexistent = 21;
  static const int  ErrNomRepetit       = 22;

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

6 La clase easy_dial

La classe `easy_dial` conté una estructura de dades que permet fer cerques usant prefixos dels noms associats als números de telèfon i usant la informació sobre la freqüència de trucades a cada un d'aquests números.

Un objecte de la classe `easy_dial` es construeix a partir de les dades registrades en un objecte de la classe `call_registry`. És important destacar que un `easy_dial` és una estructura estàtica, en el sentit que la informació sobre els telèfons no es pot modificar una cop construïda. Per tant, sobre un objecte de la classe `easy_dial` es poden fer consultes però no insercions ni esborrats.

Associat a cada objecte de la classe `easy_dial` tenim la noció de *prefix en curs*. Aquest prefix es construeix lletra a lletra i seqüencialment. L'operació `seguent` afegeix una lletra al prefix en curs pel final, i l'operació `anterior` esborra l'última lletra del prefix en curs. Per indicar fi de paraula s'usa la constant de classe `phone::ENDPREF`. Recordeu que aquesta constant és el caràcter '$\backslash 0$' el codi ASCII de la qual és 0 i per tant és el menor dels caràcters. Resulta convenient imaginar que tots els noms que hi ha en un `easy_dial` estan marcats al final amb `phone::ENDPREF`. D'aquesta manera, cap nom és prefix d'un altre i per això quan `phone::ENDPREF` es dóna com argument de l'operació `seguent` estem indicant que el prefix en curs ja és un nom exacte.

Per poder descriure amb tota precisió el comportament de les diverses operacions d'aquesta classe ens calen les següents definicions:

1. Donat un conjunt de telèfons $S = \{t_1, t_2, \ldots, t_n\}$ i una cadena p (eventualment buida)
$$\text{Pref}(S, p) = \{t \in S \mid p \text{ és un prefix del nom de } t\}.$$
Per exemple, pel conjunt de telèfons S que s'usa en la figura 1,
   - Pref(S, "MA") $= \{t_1, t_3, t_5\}$, és a dir, els telèfons de MARIA, MAR i MARTA, respectivament.
   - Pref(S, "MAR") $= \{t_1, t_3, t_5\}$.
   - Pref(S, "MAR$\backslash$0") $= \{t_3\}$.

2. Donat un conjunt de telèfons S i una cadena p (eventualment buida),
$$F(S, p) = \max \left\{ t \mid t \in \left(\text{Pref}(S, p) \setminus \bigcup_{q \subset p} F(S, q) \right) \right\},$$
a on $q \subset p$ significa que q és prefix de p (però no igual a p). El màxim del conjunt de telèfons es defineix a partir del criteri d'ordenació donat en la classe `phone`; és a dir, per freqüència, i en cas d'empat, per nom. Si el conjunt de telèfons sobre el que s'aplica $\max$ és buit llavors direm que $F(S, p)$ no existeix.

En paraules, $F(S, p)$ és el màxim telèfon el nom de qual comença pel prefix p, però no és el màxim telèfon el nom del qual comença pel prefix q, per cap prefix estricte q de p.

Decisions sobre les dades: La classe `easy_dial` és un conjunt de mida no acotada amb la propietat que tots els noms i tots els números de telèfon són diferents entre sí. A més no hi ha cap nom buit. Però l'string buit es retorna per indicar que no existeixen telèfons amb un prefix donat en l'operació `seguent`.

Implementació: La representació d'aquesta classe es trobarà en el fitxer `easy_dial.rep` i la implementació en el fitxer `easy_dial.cpp`. Òbviament s'utilitza la classe `string`.

Les operacions d'aquesta classe han de ser totes eficients en el cas pitjor o mitjà, tenint en compte que l'alfabet sobre el qual es construeixen els strings té un número constant de lletres (unes quantes desenes). En particular, és obligatori que el constructor per còpia, l'assignació i el destructor tinguin cost lineal respecte el número de noms emmagatzemats en el `easy_dial`. És obligatori també que les operacions `inici`, `seguent` i `anterior` tinguin cost constant respecte el número de noms. Finalment, l'operació consultora `num_telf` també ha de tenir cost constant.

En la implementació d'aquesta classe s'ha de tenir en compte (com a totes les classes i mòduls, però aquí especialment) el cost en espai, ja que és fàcil dissenyar estructures que consumeixin molta més memòria de la necessària.

N.B. Es valorarà especialment que es compleixin els requeriments sobre el cost temporal de les operacions d'aquesta classe.

```
#ifndef _EASY_DIAL_HPP
#define _EASY_DIAL_HPP

#include "call_registry.hpp"
#include <esin/error>
#include <esin/util>
#include <string>

using namespace std;
using util::nat;

class easy_dial {
public:
  /* Construeix un easy_dial a partir de la
  informació continguda en el call_registry donat. El
  prefix en curs queda indefinit. */
  easy_dial(const call_registry& R) throw(error);

  /* Tres grans. Constructor per còpia, operador d'assignació
  i destructor. */
  easy_dial(const easy_dial& D) throw(error);
```

```
easy_dial& operator=(const easy_dial& D) throw(error);
~easy_dial() throw();

/* Inicialitza el prefix en curs a buit. Retorna el nom de F(S, '');
si F (S, '') no existeix llavors retorna l'string buit. */
string inici() throw();

/* Retorna el nom de F(S, p') on p' és el prefix resultant d'afegir
el caràcter c al final del prefix en curs p i
fa que el nou prefix en curs sigui p'.
Si F(S, p) existeix però F(S, p') no existeix llavors retorna
l'string buit.
Si no existeix F(S, p) (i per tant tampoc pot existir F(S, p'))
llavors es produeix un error i el prefix en curs queda indefinit.
Naturalment, es produeix un error si el prefix en curs inicial p
fos indefinit. */
string seguent(char c) throw(error);

/* Elimina l'últim caràcter del prefix en curs p = p' · a
(a és el caràcter eliminat). Retorna el nom F(S, p')
i fa que el nou prefix en curs sigui p'.
Es produeix un error si p fos buida i si es fa que el prefix en curs
quedi indefinit. Òbviament, també es produeix un error
si p fos indefinit. */
string anterior() throw(error);

/* Retorna el número de telèfon de F(S, p), sent p
el prefix en curs. Es produeix un error si p és indefinit o si
no existeix F(S, p). */
nat num_telf() const throw(error);

/* Retorna en el vector result tots els noms dels contactes de
telèfon que comencen amb el prefix pref, en ordre lexicogràfic
creixent. */
void comencen(const string& pref,
              vector<string>& result) const throw(error);

/* Retorna el número mitjà de pulsacions necessàries para obtenir un
telèfon. Formalment, si X és el conjunt de noms emmagatzemats en
el easy_dial i t(s) és el número de pulsacions mínimes
necessàries (= número de crides a l'operació seguent) per
obtenir el telèfon el nom del qual és s. La funció retorna la suma
    Pr(s) · t(s)
per tots els telèfons s del conjunt X, sent Pr(s) la probabilitat de
telefonar a s. La probabilitat s'obté dividint la freqüència de s per
la suma de totes les freqüències. */
double longitud_mitjana() const throw();
```

```
  /* Gestió d'errors. */
  static const int  ErrPrefixIndef        = 31;
  static const int  ErrNoExisteixTelefon  = 32;
  static const int  ErrNoHiHaAnterior     = 33;

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

7 El mòdul dialog

Conté la funció `dialog` que simula la interacció entre el sistema `EasyDial` i un usuari. L'operació rep un objecte de la classe `easy_dial` i un string que és la seqüència de pulsacions que efectua l'usuari. A més dels caràcters convencionals l'string pot contenir aparicions del caràcter `phone::DELETECHAR='<'` que representa a la tecla d'esborrat del caràcter premut prèviament, i aparicions del caràcter `phone::ENDCHAR='|'` que l'usuari utilitza per indicar que el prefix entrat fins el moment és un nom complet.

L'operació retorna un vector d'strings amb els successius noms que el sistema presentaria per pantalla i el número de telèfon corresponent a l'últim nom o un 0 si el nom no està.

Implementació: La implementació de l'operació `dialog` i les funcions auxiliars que es considerin oportunes, es trobaran exclusivament en el fitxer `dialog.cpp`. No es pot usar cap classe de la STL (*Standard Template Library*) per la implementació d'aquest mòdul, llevat de la classe `vector`.

```
#ifndef _DIALOG_HPP
#define _DIALOG_HPP

#include "easy_dial.hpp"
#include <esin/error>
#include <esin/util>
#include <string>
#include <vector>

using namespace std;
using util::nat;

namespace dialog {

  /* Retorna en el vector answers els resultats obtinguts al processar
  els successius caràcters de l'string input, i en numtelf retorna
  el número de telèfon corresponent a l'últim nom obtingut o un 0 si
  no existeix aquest nom de telèfon. Si durant el processament de la
  seqüència de tecles representada en input es produís un error
  llavors a answers es registra el missatge d'error associat a
  l'excepció, numtelf és un 0 i finalitza el procés. */
  void dialog(easy_dial& easy, const string& input,
              vector<string>& answers, nat& numtelf) throw();

};
#endif
```

8 Errors

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

```
11 phone Nom incorrecte.

21 call_registry Numero inexistent.
22 call_registry Nom repetit.

31 easy_dial Prefix en curs indefinit.
32 easy_dial No hi ha telefons amb el prefix en curs.
33 easy_dial No es pot eliminar caracter en el prefix en curs.
```

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
```