Projecte d'ESIN
Normativa i Enunciat

Tardor de 2024

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 `jordi.esteve@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++-13.2.0`) 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:

   $$P = 0.5E + 0.5D$$

   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 12 de gener de 2025 a les 23h del vespre. 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

Avui en dia els telèfons mòbils intel·ligents són un aparell més dins la nostra vida. I els utilitzem no només per trucar o enviar missatges sinó també per navegar per internet, com a càmera de fotos o per jugar. La clau de l'èxit dels telèfons intel·ligents radica en les seves funcionalitats que es poden configurar al gust de l'usuari afegint petites aplicacions (les anomenades apps).

L'objectiu d'aquest projecte és implementar un conjunt de classes i mòduls que permetin jugar a jocs de paraules com Apalabrados, Word twist, Word challenge, Scrabble, Penjat i similars, que s'han tornat a posar de moda gràcies a les seves versions per a telèfons intel·ligents. En aquests jocs hem de formar paraules usant les lletres que ens proporcionen, normalment en un tauler dividit en caselles i amb algunes caselles especials amb puntuació extra.

En aquest tipus de jocs les estructures de dades juguen un paper fonamental doncs cal evitar càlculs complexos durant l'execució del joc. En aquest projecte ens centrarem justament en aquesta part, i no farem les parts corresponents a la interfície visual, control de temps, càlcul de la puntuació, ... que també són molt interessants i per suposat essencials per tenir un joc completament funcional.

El joc que implementarem en aquest projecte és el *Word challenge*. És un joc de paraules l'objectiu del qual és que el jugador tracti de crear, en un temps limitat, el màxim nombre de paraules vàlides possible (d'acord amb un diccionari) amb les lletres que ens han donat. El conjunt de lletres inicial està col·locat en un faristol i pot haver lletres repetides. Cada lletra del faristol només es pot emprar una vegada en cada paraula que formem. En general, el faristol conté N lletres i la longitud de les paraules vàlides varia entre 3 i N. Amb cada paraula formada s'atorguen punts i quant més llarga sigui la paraula formada més punts s'obtenen.

El diccionari juga un paper primordial donat que és el que indica quines paraules són vàlides i mitjançant aquest es determina quantes paraules es poden formar variant la posició i el nombre de lletres que es prenen del faristol.

El projecte consistirà en implementar una sèrie de mòduls i classes amb les quals es pugui resoldre eficientment el problema de trobar totes les paraules d'un diccionari que es puguin formar prenent un cert nombre de lletres —com a mínim tres— i permutant-les adequadament, d'entre les lletres d'un conjunt que se'ns dóna inicialment.

En una possible implementació del joc, el programa triaria secretament una paraula de longitud N del diccionari (per garantir que com a mínim es pot formar una paraula amb totes les seves lletres!) i usaria el mòdul que implementareu per obtenir la llista de totes les paraules que es poden formar amb les lletres de la paraula triada inicialment. A continuació es mostraria al jugador les N lletres de la paraula triada però desordenades i, de manera més o menys gràfica, la informació relativa a quantes paraules hi ha en la llista de cada longitud entre 3 i N. Llavors es posa en marxa el compte enrere: el jugador

ha de trobar tantes paraules com pugui en un temps limitat.

Òbviament, el resultat sempre dependrà del diccionari utilitzat.

Per acabar d'aclarir el joc us presentem dos exemples:

Exemple 1: Si en el faristol tenim les següents lletres:

O C A O R R

i el diccionari contingués totes les paraules del castellà, podríem formar: 8 paraules de tres lletres (ROA, OCA, ORA, ORO, ARO, ...), 13 paraules de quatre lletres (ROCA, CORO, ORCA, ARCO, CARO, ...), 5 paraules de cinc lletres (CORRA, CORRO, CARRO, ...) i 2 paraules amb les sis lletres (CORROA i ACORRO).

Exemple 2: Si el conjunt inicial de lletres en el faristol és:

T S R A O C

i el diccionari contingués totes les paraules del castellà, el sistema ens permetrà obtenir eficientment la següent llista de paraules (ordenades per longitud creixent, i a igual longitud, per ordre alfabètic creixent):
ARO, ASO, ATO, OCA, ORA, OSA, RAS, TAS, TOS, ACTO, ASCO, CARO, CASO, CATO, COSA, COTA, OCAS, ORAS, OSAR, OTRA, RASO, RATO, ROSA, ROTA, SACO, SOTA, TACO, TARO, TASO, TOCA, TOSA, TRAS, ACTOS, CAROS, CASTO, CORSA, CORTA, COSTA, COTAS, OTRAS, RATOS, ROCAS, ROTAS, TACOS, TOCAS, TOSAS, TOSCA, TROCA, CORTAS, COSTAR, COSTRA, TROCAS.

3 Disseny modular

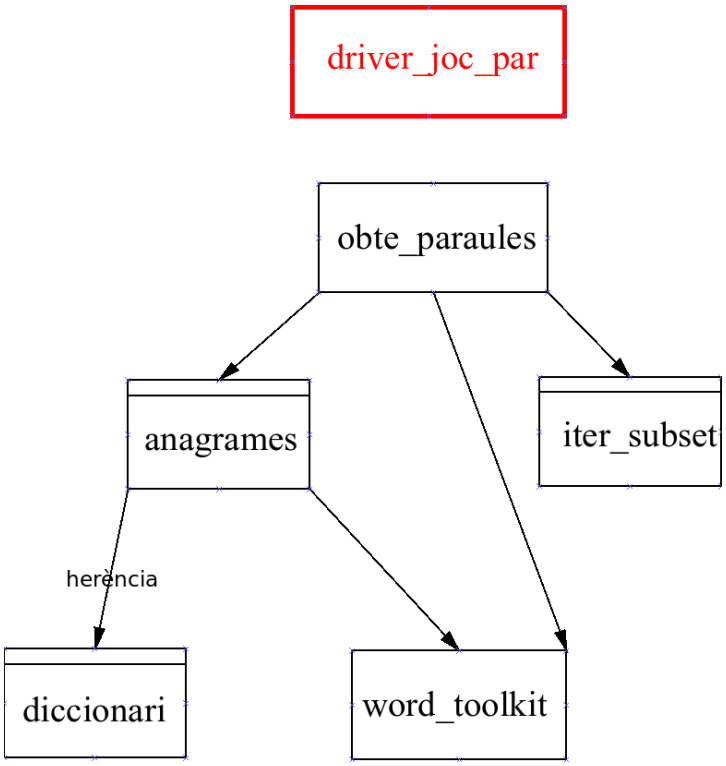

Figura 1: Disseny modular del projecte.

En aquest projecte la vostra tasca serà la de dissenyar i implementar el mòdul word_toolkit, les classes diccionari, anagrames, iter_subset, i el mòdul obte_paraules.

S'han omès d'aquest diagrama (figura 1) 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_joc_par`. 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 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 una cop hem implementat cadascuna de les classes, les sotmeteu als vostres jocs de prova. 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 el mòdul word_toolkit
- Implementar la classe iter_subset
- Implementar el mòdul obte_paraules
- Implementar la classe diccionari
- Implementar la classe anagrames

4 El mòdul word_toolkit

Aquest mòdul ofereix funcions útils per les altres classes i mòduls.

Organització de la implementació:

La implementació d'aquest mòdul es trobarà en el fitxer `word_toolkit.cpp`.

Nota: En la implementació d'aquest mòdul es pot usar el mètode sort de la biblioteca STL.

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

using namespace std;

namespace word_toolkit {

  /* Pre:  Cert
     Post: Retorna cert si i només si les lletres de l'string s estan
     en ordre lexicogràfic ascendent. */
  bool es_canonic(const string& s) throw();

  /* Pre:  Cert
     Post: Retorna l'anagrama canònic de l'string s.
     Veure la classe anagrames per saber la definició d'anagrama canònic. */
  string anagrama_canonic(const string& s) throw();

  /* Pre: L és una llista no buida de paraules formades exclusivament
     amb lletres majúscules de la 'A' a la 'Z' (excloses la 'Ñ', 'Ç',
     majúscules accentuades, ...).
     Post: Retorna el caràcter que no apareix a l'string excl i és
     el més freqüent en la llista de paraules L.
     En cas d'empat, es retornaria el caràcter alfabèticament menor.
     Si l'string excl inclou totes les lletres de la 'A' a la 'Z' es
     retorna el caràcter '\0', és a dir, el caràcter de codi ASCII 0. */
  char mes_frequent(const string& excl, const list<string>& L) throw();
};
#endif
```

5 La classe diccionari

Un objecte de la classe diccionari emmagatzema un conjunt de paraules (representades mitjançant strings) i ofereix operacions eficients per afegir noves paraules, determinar si una paraula és present en el diccionari o no, per trobar totes les paraules que compleixen un patró, etc.

Decisions sobre les dades:

Totes les paraules del diccionari estaran constituïdes exclusivament per lletres majúscules[1], de l'A a la Z. El nombre de paraules que emmagatzemarà un diccionari no es coneix ni es pot fer cap estimació raonable sobre el seu valor. Pot haver diccionaris petits, mitjans, grans i gegantins.

Comentaris i exemples sobre l'especificació:

L'especificació de les operacions constructores és senzilla i no semblen requerir més explicacions. Malgrat això, és convenient fer alguns comentaris i donar exemples que ajudin a aclarir l'especificació de les operacions `prefix` i `satisfan_patro`.

`prefix`

Si la paraula donada p està en el diccionari llavors l'operació `prefix` ens retorna la paraula sencera. I a la inversa, si `prefix` no ens retorna la paraula donada, llavors aquesta paraula no forma part del diccionari.

En canvi si p no és una paraula del diccionari ni tampoc ho és cap dels seus prefixos, llavors l'operació `prefix` retornarà l'string buit (que sempre és una paraula del diccionari).

Suposem que p = `PODARFOLGZ`. Si el diccionari conté paraules vàlides en castellà llavors l'operació `prefix` retornarà `PODAR`, a menys que aquest verb no aparegués en el diccionari, en aquest cas retornaria `PODA`. Si tampoc estigués aquesta última paraula en el diccionari, el més probable és que es retornés l'string buit doncs, en principi, ni `POD`, ni `PO` ni `P` són paraules vàlides.

Ara bé, el diccionari pot contenir paraules qualsevols i el fet que aquestes paraules siguin o no vàlides en un cert idioma és completament irrellevant des del punt de vista del disseny i implementació de la classe diccionari. Si per qualsevol raó haguéssim afegit l'string `PODARFO` en el nostre `diccionari`, llavors `prefix`("`PODARFOLGZ`") retornaria `PODARFO`. De manera anàloga si el diccionari contingués l'string `POD` (però

[1] Sense incloure ni Ñ ni Ç ni vocals accentuades; en definitiva, només les lletres majúscules amb codis ASCII entre 65 i 90.

no les paraules PODA o PODAR ni cap string més llarg) llavors prefix("PODARFOLGZ") retornaria POD.

satisfan_patro

L'operació satisfan_patro rep un patró a través d'un vector de strings i retorna una llista ordenada amb les paraules del diccionari que satisfan aquest patró. En un patró cada component és un string que indica quines lletres (de l'A a la Z) s'admeten en la posició corresponent.

Per exemple, suposem que el patró q és ("ABC...Z","C","AEIOU","LRS"). Llavors satisfan_patro(q) retornarà totes les paraules del diccionari de quatre lletres on la primera lletra és una lletra qualsevol, la segona lletra és una C, la tercera és una vocal i la quarta és L, R o S, p.e., entre altres, la paraula ECOS.

Per altra banda, amb el patró ("A...Z","A...Z","A...Z","A...Z","A...Z") retornarà totes les paraules de longitud 5 i el patró ("A","S","O") retornarà una llista que contindrà la paraula ASO si aquesta paraula estigués present o una llista buida en cas contrari. En general, l'operació retornarà una llista buida si en el diccionari no hi ha cap paraula que satisfaci el patró donat.

Organització de la representació i implementació:

La representació d'aquesta classe es trobarà en el fitxer diccionari.rep i la implementació en el fitxer diccionari.cpp.

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

using namespace std;
using util::nat;

class diccionari {

public:
  /* Pre:  Cert
     Post: Construeix un diccionari que conté únicament una paraula:
     la paraula buida. */
  diccionari() throw(error);
```

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

  /* Pre:  Cert
     Post: Afegeix la paraula p al diccionari; si la paraula p ja
     formava part del diccionari, l'operació no té cap efecte. */
  void insereix(const string& p) throw(error);

  /* Pre:  Cert
     Post: Retorna el prefix més llarg de p que és una paraula que
     pertany al diccionari, o dit d'una forma més precisa, retorna la
     paraula més llarga del diccionari que és prefix de p. */
  string prefix(const string& p) const throw(error);

  /* Pre:  Cert
     Post: Retorna la llista de paraules del diccionari que satisfan
     el patró especificat en el vector d'strings q, en ordre
     alfabètic ascendent. */
  void satisfan_patro(const vector<string>& q,
                      list<string>& L) const throw(error);

  /* Pre:  Cert
     Post: Retorna una llista amb totes les paraules del diccionari
     de longitud major o igual a k en ordre alfabètic ascendent. */
  void llista_paraules(nat k, list<string>& L) const throw(error);

  /* Pre:  Cert
     Post: Retorna el nombre de paraules en el diccionari. */
  nat num_pal() const throw();

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

6 La classe anagrames

Un objecte de la classe anagrames és un diccionari (per tant emmagatzema paraules i té operacions eficients per afegir noves paraules, determinar si una paraula és present o no, per trobar les paraules que compleixen un patró, ...), però a més a més permet que sigui eficient trobar el subconjunt de totes les paraules que tenen un mateix *anagrama canònic* donat.

Donades dues paraules p i q d'un diccionari, es diu que p és un anagrama de q (i viceversa) si p es pot obtenir permutant les lletres de q, p.e. `ACCION` és un anagrama de `COCINA`. Donada una paraula qualsevol p, anomenarem anagrama canònic `anagrama_canonic`(p) a l'string que s'obté a l'ordenar les lletres de p de menor a major, p.e., `anagrama_canonic`(`COCINA`)=`ACCINO`. És important observar que l'anagrama canònic d'una paraula p no necessàriament és una paraula vàlida del diccionari —i per tant, parlant amb propietat, no seria un anagrama.

Aquesta operació fonamental de la classe anagrames és `mateix_anagrama_canonic`(a), que retorna totes les paraules que tenen com anagrama canònic a l'string a. P.e. les paraules `PERA`, `PARE` i `RAPE` tenen com anagrama canònic l'string `AEPR`. L'operació ha de ser el més eficient possible; el seu cost ha de ser proporcional a la longitud de la llista de paraules retornada com a resultat, més un cost addicional $O(\log n)$ per trobar l'string a.

Comentaris sobre la implementació:

Aprofitarem els mecanismes d'herència que proporciona el llenguatge C++i definirem la classe anagrames com una herència pública de la la classe diccionari; així dins de la classe anagrames disposarem de tots els mètodes de la la classe diccionari. Redefinirem el mètode `insereix` per poder classificar la paraula segons el seu *anagrama canònic* a més d'afegir-la al diccionari. I afegirem el mètode `mateix_anagrama_canonic` que la classe base diccionari no tenia.

Es poden emprar les classes `vector`, `list` i els algorismes associats de la biblioteca STL en la implementació d'aquesta classe (tant en la representació com en la implementació dels mètodes). Si s'utilitzen aquestes estructures també s'haurà de justificar el perquè i calcular-ne l'eficiència en temps i espai que s'obtingui.

Organització de la representació i implementació:

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

```
#ifndef _ANAGRAMES_HPP
#define _ANAGRAMES_HPP
#include <string>
#include <list>
#include <esin/error>
#include "diccionari.hpp"
```

```
using namespace std;

class anagrames : public diccionari {

public:
  /* Pre:  Cert
     Post: Construeix un anagrama buit. */
  anagrames() throw(error);

  /* Tres grans. Constructor per còpia, operador d'assignació i destructor. */
  anagrames(const anagrames& A) throw(error);
  anagrames& operator=(const anagrames& A) throw(error);
  ~anagrames() throw();

  /* Pre:  Cert
     Post: Afegeix la paraula p a l'anagrama; si la paraula p ja
     formava part de l'anagrama, l'operació no té cap efecte. */
  void insereix(const string& p) throw(error);

  /* Pre:  Cert
     Post: Retorna la llista de paraules p tals que anagrama_canonic(p)=a.
     Llança un error si les lletres de a no estan en ordre ascendent. */
  void mateix_anagrama_canonic(const string& a, list<string>& L) const throw(error)

  /* Gestió d'errors. */
  static const int NoEsCanonic = 21;

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

7 La classe iter_subset

Un objecte de la classe iter_subset és un iterador sobre el conjunt dels subconjunts de k elements presos de $\{1, \ldots, n\}$. Un subconjunt es representarà mitjançant un `vector<nat>`. Els subconjunts es recorren en ordre lexicogràfic ascendent. Per exemple

```
iter_subset C(4,2);
while (!C.end()) {
  print(*C);
  ++C;
};
```

imprimeix tots els subconjunts de dos elements presos de $\{1, 2, 3, 4\}$; en concret s'imprimeix $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 3\}$, $\{2, 4\}$ y $\{3, 4\}$, per aquest ordre.

El cost de recórrer els $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ subconjunts de k elements presos de $\{1, \ldots, n\}$ ha de ser $\Theta\left(\binom{n}{k}\right)$.

Organització de la representació i implementació:

La representació d'aquesta classe es trobarà en el fitxer `iter_subset.rep` i la seva implementació en el fitxer `iter_subset.cpp`.

```
#ifndef _ITER_SUBSET_HPP
#define _ITER_SUBSET_HPP
#include <esin/error>
#include <esin/util>
#include <vector>

using namespace std;
using util::nat;

typedef vector<nat> subset;

class iter_subset {

public:
  /* Pre:  Cert
     Post: Construeix un iterador sobre els subconjunts de k elements
     de {1, ..., n}; si k > n no hi ha res a recórrer. */
  iter_subset(nat n, nat k) throw(error);

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

```
  /* Pre:  Cert
     Post: Retorna cert si l'iterador ja ha visitat tots els subconjunts
     de k elements presos d'entre n; o dit d'una altra forma, retorna
     cert quan l'iterador apunta a un subconjunt sentinella fictici
     que queda a continuació de l'últim subconjunt vàlid. */
  bool end() const throw();

  /* Operador de desreferència.
     Pre:  Cert
     Post: Retorna el subconjunt apuntat per l'iterador;
     llança un error si l'iterador apunta al sentinella. */
  subset operator*() const throw(error);

  /* Operador de preincrement.
     Pre:  Cert
     Post: Avança l'iterador al següent subconjunt en la seqüència i el retorna;
     no es produeix l'avançament si l'iterador ja apuntava al sentinella. */
  iter_subset& operator++() throw();

  /* Operador de postincrement.
     Pre:  Cert
     Post: Avança l'iterador al següent subconjunt en la seqüència i retorna el seu
     previ; no es produeix l'avançament si l'iterador ja apuntava al sentinella. */
  iter_subset operator++(int) throw();

  /* Operadors relacionals. */
  bool operator==(const iter_subset& c) const throw();
  bool operator!=(const iter_subset& c) const throw();

  /* Gestió d'errors. */
  static const int IterSubsetIncorr = 31;

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

8 El mòdul obte_paraules

El mòdul obte_paraules ofereix dues operacions (amb el mateix nom); una d'elles ens retorna totes les paraules que es poden formar usant un cert nombre de lletres k preses d'un cert string; l'altra ens retorna totes les paraules que es poden formar usant 3 o més lletres preses d'un string donat.

Organització de la implementació:

La implementació d'aquest mòdul es trobarà en el fitxer `obte_paraules.cpp`.

```
#ifndef _OBTE_PARAULES_HPP
#define _OBTE_PARAULES_HPP
#include <list>
#include <string>
#include <esin/error>
#include <esin/util>
#include "anagrames.hpp"

using namespace std;
using util::nat;

namespace obte_paraules {

  /* Pre:  Cert
     Post: Retorna la llista de paraules que es poden formar usant k lletres
     de la paraula s. Llança error si k és major que la longitud de
     l'string s o k < 3. */
  void obte_paraules(nat k, const string& s, const anagrames& A,
                     list<string>& paraules) throw(error);

  /* Pre:  Cert
     Post: Retorna la llista de paraules que es poden formar usant 3 o més llet
     de la paraula s. La llista estarà ordenada de menys a més longitud;
     a igual longitud les paraules estan en ordre alfabètic creixent.
     Llança un error si l'string s té menys de tres lletres. */
  void obte_paraules(const string& s, const anagrames& A,
                     list<string>& paraules) throw(error);

  /* Gestió d'errors. */
  static const int LongitudInvalida = 41;
};
#endif
```

9 Errors

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

```
21 anagrames     L'string no es un anagrama.
31 iter_subset   Acces amb iterador de subconjunts invalid.
41 obte_paraules Longitud invalida.
```

10 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
```