Pràctica de PS
Normativa i Enunciat

Tardor de 2004

Aquest document és llarg però és imprescindible que ho llegiu íntegrament i amb deteniment, àdhuc si sou repetidors, ja que es donen les instruccions i normas que heu de seguir per a que la vostra pràctica sigui avaluada positivament. El professorat de l'assignatura donarà per fet que tots els alumnes coneixeno el contingut íntegre d'aquest document.

Continguts:

1 Normativa

1. Tal i com s'explica a la Guia Docent, per a assolir els objetius de l'assignatura es considera imprescindible el desenvolupament per part de l'estudiant d'una pràctica que requereix algunes hores adicionals de treball personal (apart de les classes de laboratori, ón es fa el desenvolupament dels altres exercicis pràctics i us permetran familiaritzar-vos amb l'entorn de treball i el llenguatge de programació C++).

2. La pràctica es realitzarà en equips de dos estudiants. Si un d'ells abandona, un dels integrants haurà de notificar-lo amb la màxima prontitud via e-mail a `ps@fib.upc.es`, i continuar la pràctica en solitari, eventualment. Per altra banda, solament es permetrà la formació d'equips individuals en casos excepcionals ón sigui impossible reunir-se o comunicar-se amb altres estudiants, i s'haurà de justificar mitjançant algún tipus de document. Consulteu l'apartat *Equips de Pràctiques* a les pàgines Web de l'assignatura (`http://www.fib.upc.es/~ps`), ón es donen les instruccions sobre la formació d'equips.

3. El soport que fareu servir per aquesta pràctica és el llenguatge de programació C++ (específicament el compilador GNU `gcc-2.95.2`) sobre l'entorn Solaris (`moonrey`) del LCFIB. 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 de la pràctica des del vostre equip particular a l'entorn del LCFIB, 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 al menys una comprovació final de que el programa desenvolupat en PC o similar funcioni efectivament en l'entorn Solaris (`moonrey`) del LCFIB.

4. La pràctica serà avaluada mitjançant:

   - la seva execució en l'entorn del LCFIB amb una sèrie de *jocs de prova* i
   - la correcció del disseny, implementació i documentació: les decisions de disseny i la seva justificació, la eficiència dels algorismes i estructures de dades, la legibilitat, robustesa i estil de programació, etc. Tota la documentació ha de acompanyar el codi; no heu d'entregar cap documentació en paper.

   Existeixen dos tipus de jocs de prova: públics i privats. Els jocs de prova públics es posaran a la vostra disposició amb antel·lació suficient a les pàgines Web de l'assignatura (`http://www.fib.upc.es/~ps`).

   La nota de la pràctica es calcula a partir de la nota d'execució (E) i la nota de disseny (D). La nota de la pràctica és:

   $$P = 0.3E + 0.7D,$$

   si ambdues notes parcials (E y D) són majors que 0; $P = 0$ si qualsevol d'elles fos 0.

El document *Guía y Normas de Programación* descriu, entre altres coses, les situacions que originen una qualificació de 0 en el disseny (i per tant una qualificació de 0 de la pràctica). La nota d'execució (E) és 5 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 5 punts més, en cas que s'hagin superat els jocs de prova públics.

A la nota P calculada segons s'ha descrit abans, se l'hi aplicarà una penalització en cas que l'entrega es faci amb retard.

5. La data límit de l'entrega final és el 7 de Gener de 2005 a les 12 del migdia. Existeix un període breu posterior a la data límit en el qual s'admet l'entrega (amb penalització). Transcorregut el període esmentat, si un equip no ha entregat la seva pràctica llavors la nota serà 0. A les pàgines Web de l'assignatura (`http://www.fib.upc.es/~ps`) es donaran tots els detalls sobre el procediment d'entrega de la pràctica.

6. No subestimeu el temps que haureu de esmerçar a cadascun dels aspectes de la pràctica: disseny, codificació, depuració d'errors, proves, documentació, ...

2 Enunciat de la pràctica

En aquesta pràctica hareu de desenvolupar una sèrie de classes i mòduls per fer un sistema que generi i resolgui laberints al que anomenarem EDAMAZE.

2.1 Una mica de història

En el temps on els déus encara corrien per la terra, hi havia un home anomenat Minos que volia ser rei de Creta. Com que era fill de Zeus (soberà de tots els déus) tenia els seus favors, i per demostrar-ho va demanar a Posidó (déu dels oceans i mars) que fes sortir un toro del mar amb la promesa que tot seguit el sacrificaria en honor seu. Posidó va accedir als seus desitjos i així va ser com un esplèndid brau va eixir prodigiosament de les aigües.

D'aquesta manera Minos va aconseguir el tron de Creta, però no es va veure amb cor d'immolar un animal tan bell. El va afegir als seus ramats i va sacrificar-ne una altre de menys esplèndid. La venjança de Posidó fou terrible: féu que l'esposa de Minos, la reina Pasífae, s'enamorés follament del toro.

Pasífae es va unir amb el toro per mitjà d'una estratagema ideada i executada per Dèdal, un gran geni d'aquell temps. Va fabricar una vaca de fusta, buida per dins i recoberta d'una pell vaquina, a l'interior de la qual es va amagar la reina. Tot seguit la va deixar en un prat on solia pasturar el toro, que ben aviat la va descobrir i s'hi va aparellar. Al cap de nou mesos Pasífae va donar a llum una criatura monstruosa, amb cos humà i cap de toro. Li posaren el nom d'Asterió però fou més conegut com el Minotaure. Com és natural Minos s'adonà de seguida que no era fill seu i, ple de vergonya i horror, el volgué ocultar dels ulls de tothom. Amb tal finalitat Dèdal va dissenyar i construir el Laberint, un edifici de corredors, sales i patis tan intricat que ningú que hi entrés podia trobar la sortida. Allà va créixer i va viure el Minotaure, sol i allunyat de tot contacte humà. Només un cop a l'any entraven en el Laberint set nois i set noies que Atenes enviava com un tribut anual a Minos. Tots ells acabaven devorats pel monstre.

Atenes, doncs, havia d'escollir cada any catorze nois d'entre el seu millor jovent i contemplar amb immensa aflicció com partien cap a Creta perquè no se'n tornés a saber mai més res. Però va arribar un any que el príncep Teseu, va demanar ser inclòs a la tria amb la intenció d'acabar amb aquella sagnia.

Quan desembarcaren a Creta, el bell Teseu va enamorar al primer cop d'ull Ariadna, una de les filles de Minos i Pasífae, que es trobava entremig de la gernació que els havia anat a rebre al port. La princesa sabia quina sort esperava al jove i, desperada, va córrer a suplicar a Dèdal que ideés un pla per salvar-lo. L'inventor cedí als seus precs i li donà un cabdell de fil amb les instruccions per no perdre el camí dins el Laberint. Ariadna va fer arribar fil i instruccions a Teseu.

Figure 1: Mossaic del Laberint de Creta.

Aquesta vegada, però, tot va ser diferent: els joves atenesos van penetrar dins el Laberint però un cop dins Teseu no va permetre que perdessin la serenitat. Van endinsar-s'hi caminant lentament un bona estona, amollant el fil que li havia proporcionat Ariadna al llarg de tot el trajecte i per fi se'l van trobar. El Minotaure es va precipitar damunt seu, però Teseu s'hi va enfrontar valentament, desarmat i tot. Després d'una llarga i violenta lluita el príncep va aconseguir matar el monstre. Tot seguit van poder desfer el camí gràcies al guiatge del fil i van eixir sans i estalvis del Laberint.

Sergi Giralt[1]

2.2 El Laberint

Els laberints, en general, es poden classificar de moltes maneres: dimensió, topologia, geometria de les cel·les...Nosaltres ens centrarem en laberints ortogonals de dues dimensions. Si agafem com a mesura de classificació la relació entre les cel·les del laberint (els camins) obtenim les següents categories:

- Laberint *perfecte* és aquell que no té cicles (circuits tancats) ni àrees inaccessibles. També se l'anomena laberint simplement connectat. Des de cada punt hi ha només un únic camí a un altre punt. Aquest tipus de laberint té exactament una solució.
- Laberint *trenat* és aquell que no té camins "cul de sac" (camins sense sortida). També s'anomena com laberint purament i múltiple connectat. Aquests tipus de laberint tenen passadisos que s'enrosquen i tornen enrera, i per això pot ser que quan s'intenta ressoldre es perdi el temps fent cercles enlloc d'anar a parar a un cul de sac. Un laberint trenat ben dissenyat pot ser més difícil de resoldre que un altre perfecte de la mateixa mida.

[1] `http://www.xtec.es/~sgiralt`

- Laberint *serp* és aquell que no té bifurcacions i per tant només té un únic camí. També s'anomena laberint de camí únic. És molt fàcil de ressoldre, ja que a no ser que facis mitja volta sempre arribaràs al final.
- Laberint *trenat parcialment* és un laberint mixte amb cicles i cul de sac. Depenent de la quantitat de cicles i camins sense sortida, direm que està fortament o feblement trenat.

El nostre sistema EDAMAZE únicament generarà automàticament laberints ortogonals perfectes, però ha de ser capaç d'emmagatzemar qualsevol tipus de laberint.

2.3 Format del Laberint

El laberint es pot carregar i guardar en un fitxer de texte. El format d'aquest fitxer serà el següent:

- El fitxer es composa de línies separades per salts de línia (`\n`).
- A la primera línia sempre apareixerà el número de files i de columnes del laberint, la qual cosa ens permetrà simplificar la lectura de dades. El número de files i el de columnes està separat per un espai en blanc.
- Cadascuna de les línies següents representa les cambres del laberint, és a dir, les parets, portes o espai buit del laberint. Els caràcters vàlids per tal de representar cadascuna d'aquestes línies són: `*`, ` `[2].
- Una cambra la dibuixarem com un espai en blanc envoltat per vuit `*` que representaran les quatre parets i les quatre cantonades.

```
***                             nord
* *                        oest      est
***                              sud
```

  Si una de les parets de la cambra té una porta es representarà amb un espai en blanc (sense `*`).

```
***
*    <-- En aquest exemple la paret est té una porta.
***
```

  Cal tenir en compte que les cambres del laberint tenen parets i portes en comú.

Un exemple de laberint amb format correcte seria (el salt de línia no es representa car ja està implícit a l'hora d'escriure'l):

```
3 11
***********************
* *   * *   *     *   *
* *** * * * *** * * ***
* *   *   * *   *     *
* * * ***** * * ***** *
*   *         * *     *
***********************
```

[2] espai

2.4 Construcció del Laberint

Definició **1** *Dues C i C' cambres d'un laberint estan* comunicades *si existeix un camí que uneix a aquestes dues cambres.*

Tenint en compte l'anterior definició es pot crear crear un laberint perfecte seguint aquest procediment:

1. Cal crear tants conjunts com cambres, i a cada conjunt hi haurà una cambra diferent.

2. Sel·leccionar una cambra a l'atzar que anomenarem C, i una de les parets a l'atzar, sempre i quan aquesta paret no tingui ja una porta i no sigui una paret que doni a l'exterior.

3. Anomenem C' a la cambra adjacent a C. Si les dues cambres, C i C', que estan separades per la paret que hem triat no estan comunicades (pertanyen a 2 conjunts diferents), llavors cal comunicar-les. Hem de crear una porta a la paret indicada i unir els 2 conjunts.

4. En cas contrari, no fem res perquè ja existeix un camí per arribar d'una cambra a l'altra.

5. S'acaba quan totes les cambres del laberint estan al mateix conjunt.

Aquest algorisme és interessant perquè no crea el laberint en forma d'arbre, excava el passadisos per tot el laberint a l'atzar però al final s'obté un laberint perfecte. Es necessita espai addicional proporcional a la mida del laberint i poder enumerar cada paret.

El següent exemple mostra un laberint sense excavar:

```
***********************
* * * * * * * * * * * *
***********************
* * * * * * * * * * * *
***********************
* * * * * * * * * * * *
***********************
* * * * * * * * * * * *
***********************
```

Una possible solució després d'haver excavat el laberint seria aquesta:

```
***********************
* *   * *   *     *   *
* *** * * * *** * * ***
* *   *   * *   *     *
* * * * *** * * ***** *
* * * *   * * * *     *
* * * ***** * * ***** *
*   *         * *     *
***********************
```

3 Disseny modular

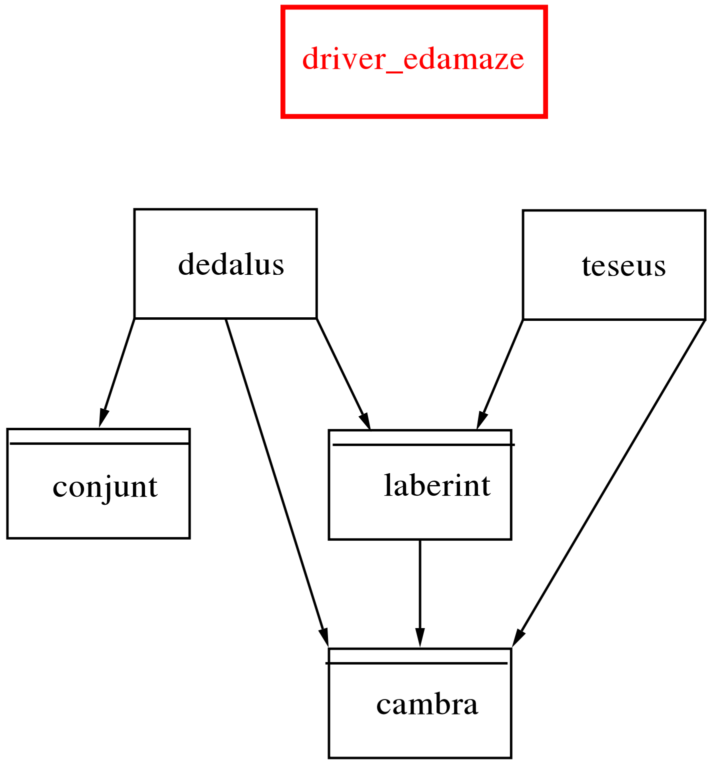

Figure 2: Disseny modular de la pràctica.

La pràctica consisteix en la implementació del disseny modular que es descriu en aquesta secció. La vostra missió serà implementar algunes de les classes i mòduls.

El mòdul `driver_edamaze` conté el programa principal i s'us dóna implementat. Permet invocar cadascuna de les operacions dels diferents mòduls i classes. Usa a tots els demés mòduls i classes encara que no s'han representat les fletxes d'ús explícitament.

A més dels mòduls i classes representades al diagrama, s'usaran les classes error i mem_din i el mòdul util de la llibreria `libeda`. Les classes error i mem_din estan documentades a la *Guía y Normas de Programación*. El mòdul util està documentat *on-line* en `<eda/util>`.

A tots els mòduls o classes d'aquesta pràctica es pot usar també la classe string de la llibreria estàndar de C++.

Recordeu que no es pot utilitzar cap classe de la llibreria externa en un dels vostres mòduls o classes, exceptuant si en aquesta documentació s'indica el contrari.

A totes les classes cal implementar els mètodes de construcció per còpia, assignació i destrucció davant la possibilitat que empreu memòria dinàmica per implementar la classe en qüestió. Si no s'usa memòria dinàmica, la implementació d'aquests tres mètodes és molt senzilla doncs només caldrà imitar el comportament dels que serien els corresponents mètodes per defecte (la destructora no fa "res", i les altres fan còpies atribut per atribut).

4 La clase paret

```
#ifndef _PARET_HPP
#define _PARET_HPP
#include <string>
```

Un objecte de la classe `paret` representa una paret d'una cambra de laberint. La proporcionem ja implementada. Un objecte d'aquesta classe pot adquirir els valors `NORD`, `EST`, `SUD`, `OEST` i `NO_DIR`. Aquest darrer valor actua com a sentinella en recorreguts sobre la seqüència de valors. També estan definits els operadors relacionals d'igualtat i desigualtat i els d'ordre (`NORD < EST < SUD < OEST < NO_DIR`).

```
class paret {
public:
  enum {NORD, EST, SUD, OEST, NO_DIR};
  paret(int p = NO_DIR) throw() : p_(p) {}
  paret(const string& s) throw() {
     p_ = NO_DIR;
     if (s == "nord" || s == "NORD" || s == "N") p_ = NORD;
     if (s == "est" || s == "EST" || s == "E")   p_ = EST;
     if (s == "sud" || s == "SUD" || s == "S")   p_ = SUD;
     if (s == "oest" || s == "OEST" || s == "W") p_ = OEST;
  }

  paret& operator++() throw() {
    p_ = (p_ == NO_DIR) ? NO_DIR : p_ + 1;
    return *this;
  }

  operator int() const throw() {
    return p_;
  }
private:
  int p_;
};
#endif
```

5 La classe cambra

Un objecte de la classe `cambra` representa una habitació rectangular del laberint, i conté informació sobre les parets que tenen porta a una altra cambra. És a dir, indica si les parets `NORD`, `SUD`, `EST` o `OEST` tenen porta o no. *Implementació:* La representació d'aquesta classe es trobarà al fitxer `cambra.rep` i la implementació al fitxer `cambra.cpp`. La documentació serà mínima, donada la senzillesa de les dades i els mètodes.

```
#ifndef _CAMBRA_HPP
#define _CAMBRA_HPP
#include <string>
#include <eda/error>
#include <eda/util>
#include "paret.hpp"

class cambra {
public:
```

Construeix una cambra amb portes als llocs indicats.

```
  explicit cambra(bool n=false, bool s=false,
                  bool e=false, bool o=false) throw(error);
```

Constructora per còpia, assignació i destructora.

```
  cambra(const cambra & c) throw(error);
  cambra& operator=(const cambra & c) throw(error);
  virtual ~cambra() throw();
```

Còpia via punters: Crea una nova cambra idèntica a la cambra sobre la qual apliquem el mètode i retorna un punter a la nova cambra.

```
virtual cambra* clone() throw(error);
```

Retorna si l'habitació a la paret indicada té una porta.

```
virtual bool hi_ha_porta(paret p) const throw();
```

Crea una nova porta a la paret indicada.

```
virtual void nova_porta(paret p) throw();
```

Consulta i actualització de les marques.

```
virtual void marca() throw() {};
```

```
virtual void marca(int n) {};

virtual bool marca_b() const throw() { return false; };
virtual int  marca_i() const throw() { return 0; };
```

Igualtat i desigualtat entre cambres. Dues cambres es consideren iguals si tenen les mateixes portes a les mateixes parets.

```
virtual bool operator==(const cambra & c) const throw();
virtual bool operator!=(const cambra & c) const throw();
```

Gestió d'errors i informació interna.

```
  static const char nom_mod[] = "cambra";

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

6 La classe cambra_marcable

```
#ifndef _CAMBRA_MARC_HPP
#define _CAMBRA_MARC_HPP
#include <string>
#include <eda/error>
#include <eda/util>
#include "cambra.hpp"
#include "paret.hpp"
```

Un objecte de la classe `cambra_marcable` representa una cambra sobre la qual podem efectuar dos tipus de marques: una marca booleana i una marca que és un número enter. *Implementació:* La representació d'aquesta classe es trobarà al fitxer `cambra_marcable.rep` i la implementació al fitxer `cambra_marcable.cpp`. La documentació serà mínima, donada la senzillesa de les dades i els mètodes.

```
class cambra_marcable : public cambra {
public:
```

Construeix una cambra amb portes als llocs indicats i amb les marques b i m donades.

```
  explicit cambra_marcable(bool b = false, int m = 0,
                   bool n=false, bool s=false,
                   bool e=false, bool o=false) throw(error);
```

Constructora per còpia, assignació i destructora.

```
  cambra_marcable(const cambra_marcable& c) throw(error);
  cambra_marcable& operator=(const cambra_marcable& c) throw(error);
  ~cambra_marcable() throw();
```

Còpia via punters: Crea una nova cambra idèntica a la cambra sobre la qual apliquem el mètode i retorna un punter a la nova cambra.

```
cambra_marcable* clone() throw(error);
```

Consulta i actualització de les marques.

```
void marca() throw();
void marca(int n) throw();

bool marca_b() const throw();
int  marca_i() const throw();
```

Gestió d'errors i informació interna.

```
  static const char nom_mod[] = "cambra_marcable";
```

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

7 La classe **laberint**

Els objectes de la classe `laberint` permeten representar i manipular laberints ortogonals explicats a la secció 2.2. Aquesta classe permetrà emmagatzemar qualsevol tipus de laberint, tant sigui perfecte, trenat,... El laberint estarà constituit per cambres d'idèntic tipus, que es determina quan el laberint es crea.

Decisions sobre les dades: La mida del laberint no està acotada. Donades dues cambres adjacents que estan comunicades per una porta (atès que les cambres del laberint comparteixen parets i portes), cadascuna d'aquesta dues cambres ha de tenir una porta en la paret que les uneix.

Implementació: La representació d'aquesta classe es trobarà en el fitxer `laberint.rep` i `iterator.rep` i la implementació en el fitxer `laberint.cpp`.

```
#ifndef _LABERINT_HPP
#define _LABERINT_HPP

#include <string>
#include <utility>
#include <eda/error>
#include <eda/util>
#include "cambra.hpp"

using util::nat;
typedef pair<nat,nat> posicio;

class laberint {
public:
```

Constructora d'un laberint buit sense excavar de la mida indicada. Totes les cambres del laberint no tenen cap porta. Es produeix un error si el número de files o de columnes del laberint no és major que 0. Totes les cambres són idèntiques a la cambra donada.

```
  explicit laberint(cambra* pc, nat num_fil=5, nat num_col=5) throw(error);
```

Constructora a partir d'un fitxer amb el nom donat s. El format del fitxer seguirà l'exposat a l'apartat 2.3. Es presuposa que el laberint és correcte. Les cambres del laberint són del mateix tipus que la cambra donada mitjançant el punter pc.

```
  explicit laberint(cambra* pc, const string& s) throw(error);
```

Constructora per còpia, assignació i destructora.

```
  laberint(const laberint & l) throw(error);
  laberint & operator=(const laberint & l) throw(error);
  ~laberint() throw(error);
```

Retornen el número de files i columnes que té el laberint, respectivament.

```
nat num_files() const throw();
nat num_columnes() const throw();
```

Retorna un punter a la cambra situada a la posició especificada per la posició *pos*. Aquest mètode permet utilitzar laberint d'aquesta manera:

`laberint l(3,4); cambra* pc; posicio pos = make_pair(2,2); l(pos); l(2,2);`

```
cambra* operator()(const posicio& pos) throw(error);
cambra* operator()(nat i, nat j) throw(error);
```

Escriu el laberint en el string s. El format de l'`string` seguirà l'exposat a l'apartat 2.3.

```
void escriu(string& s) const throw();
```

Gestió d'errors i informació interna.

```
  static const char nom_mod[] = "laberint";

  static const int FormatIncorrecte   = 31;
  static const int FilsColsIncorrecte = 32;

  static const char MsgFormatIncorrecte[]
                        = "El format de l'string es incorrecte.";
  static const char MsgFilsColsIncorrecte[]
                        = "El num de fils o cols es incorrecte.";

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

8 El mòdul teseus

Aquest mòdul defineix el *namespace* `teseus` que inclou la funció `buscar`. Aquesta funció rep un `laberint` i troba el camí més curt entre la cambra inicial i la final.

> *—Sólo hay una manera —recitó, en efecto, Guillermo— de encontrar la salida de un laberinto. Al llegar a cada nudo nuevo, o sea hasta el momento no visitado, se harán tres signos en el camino de llegada. Si se observan signos en alguno de los caminos del nudo, ello indicará que el mismo ya ha sido visitado, y entonces sólo se marcará un signo en el camino de llegada. Cuando todos los pasos de un nudo ya estén marcados, habrá que retroceder. Pero si todavía quedan uno o dos pasos sin marcar, se escogerá uno al azar, y se lo marcará con dos signos. Cuando se escoja un paso marcado con un solo signo, se marcarán dos más, para que ya tenga tres. Si al llegar a un nudo sólo se encuentran pasos marcados con tres signos, ello indicará que ya se han recorrido todas las partes del laberinto.*
>
> *—¿Cómo lo sabéis? ¿Sois experto en laberintos?*
>
> *—No, recito lo que dice un texto antiguo que leí en cierta ocasión.*
>
> *—¿Y con esas regla se puede encontrar la salida?*
>
> *—Que yo sepa, casi nunca. Pero igual probaremos.*
>
> *El nombre de la rosa,* Umberto Eco

Implementació: La implementació d'aquest mòdul es trobarà en el fitxer `teseus.cpp`.

```
#ifndef _TESEUS_HPP
#define _TESEUS_HPP
#include <eda/error>
#include <eda/util>
#include <list>
#include <string>
#include <utility>

using util::nat;

namespace teseus {
```

Obté una llista dels noms de les cambres per on passa el camí que ressol el laberint emmagatzemat al fitxer anomenat M. Forçosament el primer element d'aquesta llista serà la cambra inicial i l'última la cambra final. Es produeix un error si no hi ha cap camí que vagi des de la cambra inicial del laberint a la final, o si la cambra inicial o final no són vàlides.

```
  void buscar(const string& M,
              const posicio& ini, const posicio& final,
              list<posicio>& L) throw(error);
```

Gestió d'errors i informació interna.

```
  static const char nom_mod[] = "teseus";

  static const int SenseSolucio   = 61;
  static const int IniciFiNoValid = 62;

  static const char MsgSenseSolucio[]
                        = "El laberint no te solucio";
  static const char MsgIniciFiNoValid[]
                        = "La cambra d'inici o de final no es valida.";

};
#endif
```

9 El mòdul dedalus

Aquest mòdul defineix el *namespace* `dedalus` que inclou la funció `construir`. Aquesta funció rep un `laberint` i excava el laberint fins obtenir un laberint *perfecte*, és a dir, un laberint que no té àrees inaccessibles i tampoc cicles.

Definicions: Un laberint està *excavat* si i només si alguna de les seves cambres té alguna paret amb porta.

Implementació: La implementació d'aquest mòdul es trobarà al fitxer `dedalus.cpp`. Donat que la implementacio es opcional, es pot implementar trivialment posant un cos buit. El joc de proves public no fa cap prova sobre la funcio `construir` definida per aquest modul.

```
#ifndef _DEDALUS_HPP
#define _DEDALUS_HPP

#include <stack>
#include <string>

#include <eda/error>
#include <eda/util>
#include "conjunt.hpp"
#include "laberint.hpp"

namespace dedalus {
```

Excava un laberint de manera que construeix un laberint perfecte. Es produeix un error en cas que el laberint ja hagi estat excavat previament.

```
  void construir(laberint &M) throw(error);
```

Gestió d'errors i informació interna.

```
  static const char nom_mod[] = "dedalus";

  static const int EstaExcavat = 51;

  static const char MsgEstaExcavat[]
                          = "El laberint ja esta excavat";
};
#endif
```

10 Documentació

Els fitxers entregats han d'estar degudament documentats. És molt important descriure amb detall i precissió la representació escollida, justificant les eleccions fetes, així com les operacions de cada classe. En el cas dels fitxers `.rep` y `.cpp` és especialment important explicar en detall les representacions i els motius de la seva elecció enfront a possibles alternatives, i els algorismes utilitzats. Un cop enviats els fitxers per via electrònica, aquests serán impressos per a la seva avaluació. No haureu d'imprimir-los vosaltres. No haureu d'entregar 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 < fichero.cpp > fichero.formateado
```

i a continuació podeu visualitzar el resultat amb

```
% a2ps fitxer.formatejat -o - | gv -
```

o imprimir el fitxer mitjançant

```
% a2ps -Pnom_impressora fitxer.formatejat
```

Alternativament, podeu generar un fitxer postscript:

```
% a2ps fitxer.formatejat -o fitxer.ps
% gv fitxer.ps # per a visualitzar-lo
% lpr -Pnom_impressora fitxer.ps # per a imprimir-lo
```