Pràctiques EDA
s1: STL (`vector`, `list` i iteradors)

GEINF GDDV

Departament Informàtica i Matemàtica Aplicada
Escola Politècnica Superior
Universitat de Girona

Curs 2022/23

Universitat de Girona
Departament d'Informàtica, Matemàtica Aplicada i Estadística

(PDF creat el 23 de setembre de 2022)

Taula de continguts

La STL

La Standard Template Library (STL) és una biblioteca de C++ que ofereix:

- Contenidors genèrics (`vector`, `deque`, `list`, `set`, `map`...)
- Iteradors per recórrer els contenidors (directes, inversos, aleatoris...)
- Algoritmes genèrics per aplicar als contenidors (`find`, `sort`, `merge`...)
- Altres elements (adaptadors, objectes-funció...)

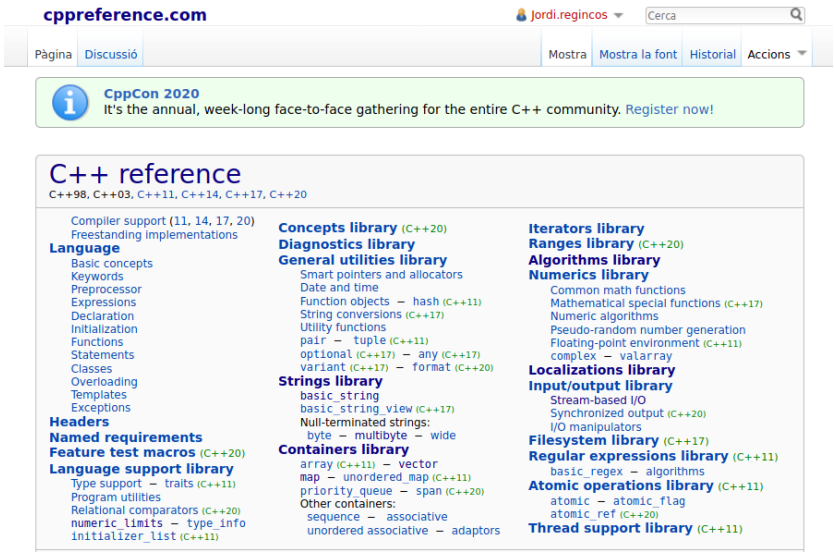
i tots aquests elements es poden anar acoblant entre ells.

La STL

La Standard Template Library (STL) permet:

- Fer abstracció de la implementació de les estructures de dades i centrar-se en l'elecció de les millors ED pels problemes concrets.
- Per cada contenidor, saber l'ordre del cost de cada operació:
  - Inserció al principi, al final o en un punt qualsevol
  - Consultar el primer element, l'últim o un de qualsevol
  - Eliminar el primer, l'últim o un de qualsevol
  - Altres operacions que només tinguin sentit en un contenidor concret

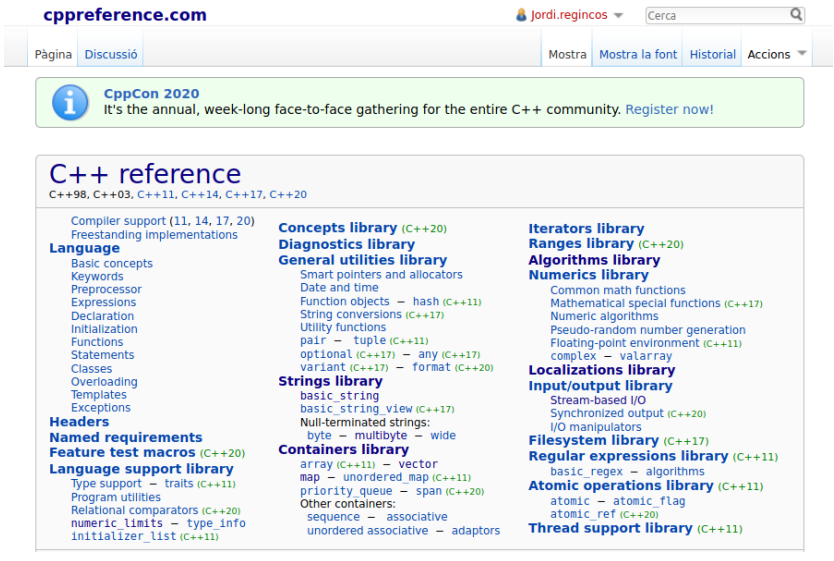
La STL: material de referència: webs

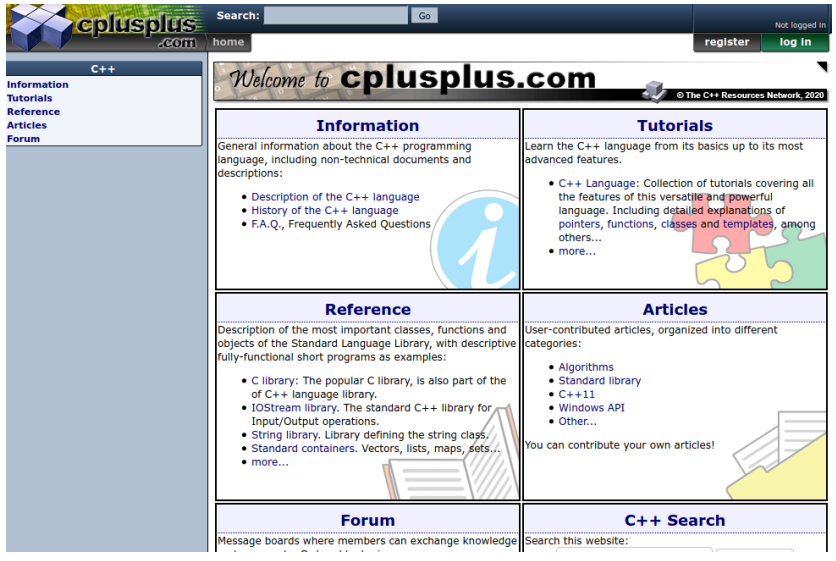


cppreference.com

http://en.cppreference.com

La STL: material de referència: webs





cppreference.com

http://en.cppreference.com

The C++ Resources Network

http://www.cplusplus.com/

Contenidors

STL ofereix diversos contenidors:

Seqüencials: `vector`, `list`, `forward_list` i `deque`

Contenidors

STL ofereix diversos contenidors:

Seqüencials: `vector`, `list`, `forward_list` i `deque`

Associatius ordenats: `map`, `multimap`, `set` i `multiset`

Contenidors

STL ofereix diversos contenidors:

Seqüencials: `vector`, `list`, `forward_list` i `deque`

Associatius ordenats: `map`, `multimap`, `set` i `multiset`

Associatius no ordenats: `unordered_map`, `unordered_multimap`, `unordered_set` i `unordered_multiset`

Contenidors

STL ofereix diversos contenidors:

Seqüencials: `vector`, `list`, `forward_list` i `deque`

Associatius ordenats: `map`, `multimap`, `set` i `multiset`

Associatius no ordenats: `unordered_map`, `unordered_multimap`, `unordered_set` i `unordered_multiset`

Adaptadors: `priority_queue`, `queue` i `stack`

Contenidors

STL ofereix diversos contenidors:

Seqüencials: `vector`, `list`, `forward_list` i `deque`

Associatius ordenats: `map`, `multimap`, `set` i `multiset`

Associatius no ordenats: `unordered_map`, `unordered_multimap`, `unordered_set` i `unordered_multiset`

Adaptadors: `priority_queue`, `queue` i `stack`

A les classes d'EDA en treballarem només uns quants:

`vector`, `list`, `map` i `set` (i algunes variants)

. . . tot i que a les pràctiques els podeu fer servir tots (tret que us demanem fer-ne servir algun de concret)

Iteradors

Els iteradors són una espècie d'apuntadors que permeten realitzar recorreguts sobre els elements dels contenidors. N'hi ha de diversos tipus, en funció del contenidor:

Input: operadors ++, ==, != i * (llegir contingut)

Output: operador ++ i * (escriure contingut un cop)

Forward: com input i output (permetent escriure diverses vegades)

Bidirectional: com forward afegint operador −−

Random: com bidireccional més operadors [], + i − (entre iterador i enter), − entre dos iteradors i <,<=,>,>=

Iteradors

Els iteradors són una espècie d'apuntadors que permeten realitzar recorreguts sobre els elements dels contenidors. N'hi ha de diversos tipus, en funció del contenidor:

Input: exemple: istream

Output: exemple: ostream

Forward: exemple: contenidor `forward_list`

Bidirectional: exemple: contenidors `list`, `map`, `set`

Random: exemple: contenidor `vector`

Iteradors

Els iteradors són una espècie d'apuntadors que permeten realitzar recorreguts sobre els elements dels contenidors. N'hi ha de diversos tipus, en funció del contenidor:

Input: exemple: istream

Output: exemple: ostream

Forward: exemple: contenidor `forward_list`

Bidirectional: exemple: contenidors `list`, `map`, `set`

Random: exemple: contenidor `vector`

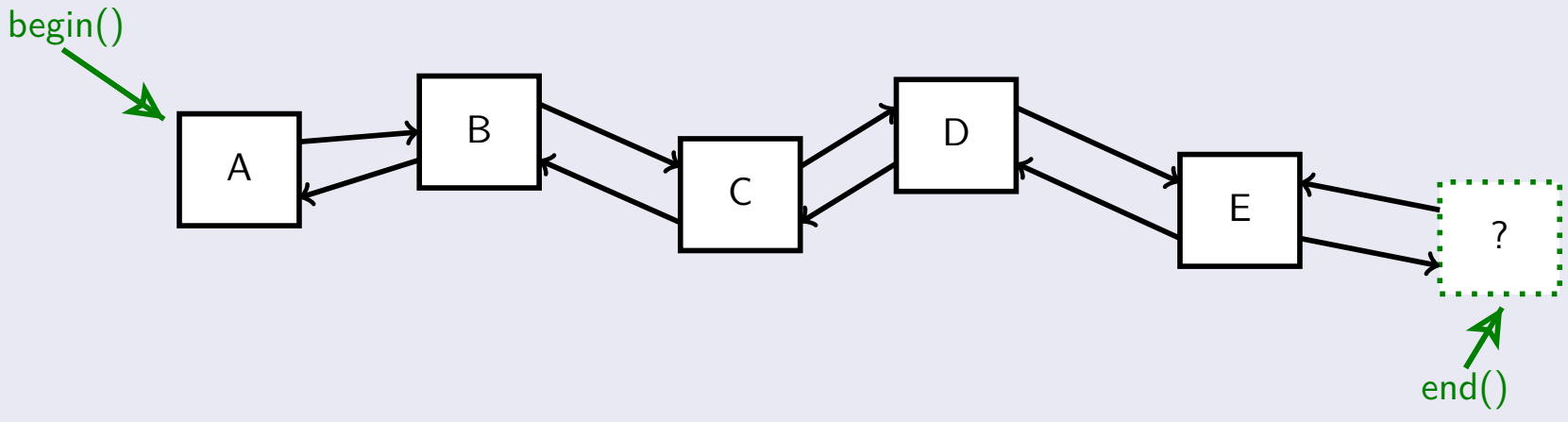
Avui en veurem dos exemples

- Bidireccional quan expliquem el contenidor `list`
- Random quan expliquem el contenidor `vector`

Iteradors

Una seqüència està definida pels elements del contenidor dins el rang `[begin,end)`, essent `begin` i `end` dos iteradors.

Exemple rang `iterator` sobre un contenidor sencer

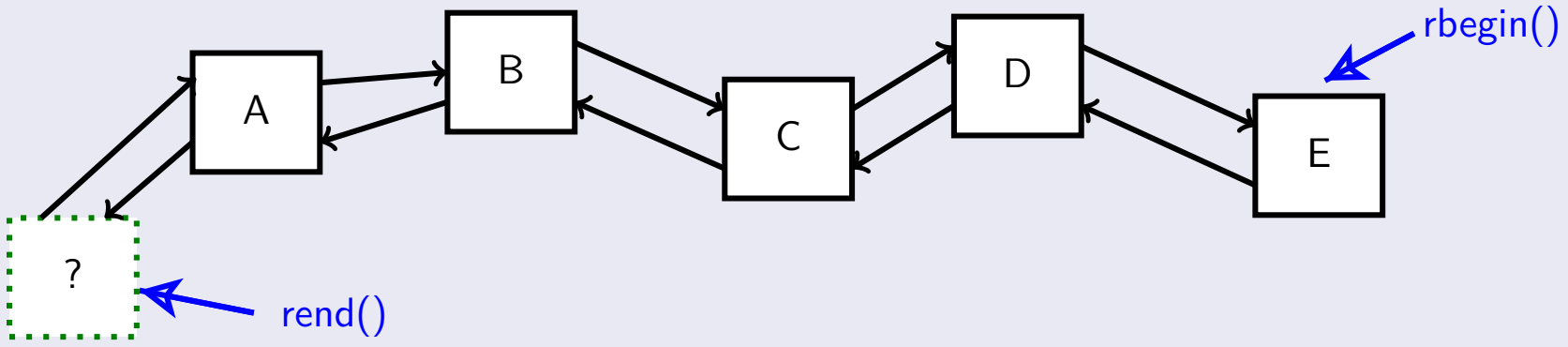


Si `C` és un contenidor llavors `C.begin()` apunta al primer element del contenidor i `C.end()` apunta a la posició després de l'últim. Tots dos són `iterators`. És un error accedir al contingut de `C.end()`.

Iteradors

Una seqüència està definida pels elements del contenidor dins el rang `[begin,end)`, essent `begin` i `end` dos iteradors.

Exemple rang `reverse_iterator` sobre un contenidor sencer



Si `C` és un contenidor llavors `C.rbegin()` apunta a l'últim element del contenidor i `C.rend()` apunta a la posició anterior al primer. Tots dos són `reverse_iterators`. És un error accedir al contingut de `C.rend()`.

Iteradors

Iteradors constants

Si necessitem utilitzar iteradors en mètodes `const` necessitarem una especialització dels iteradors que garanteixen que les seves operacions no modifiquen per res el contenidor. Són el `const_iterator` i el `const_reverse_iterator`.

Algoritmes

Més endavant...

Els algoritmes els presentarem durant les properes classes...

Taula de continguts

El contenidor `vector`

El contenidor `vector` el podem veure com una taula del C++ molt més flexible que:

- Ofereix accés directe als seus elements (cost constant).
- Permet insercions i eliminacions al final (cost constant amortitzat).
- Permet insercions i eliminacions pel mig (cost lineal).
- No té limitacions de mida (s'expandeix de manera transparent).
- Té associats iteradors aleatoris (Random), que permeten ++, −−, +=, −=, accés directe element n ...

Notació simplificada

Notació

En tot el material farem servir una notació simplificada de les declaracions de les STL amagant les complexitats sintàctiques dels *templates*.

- Farem servir `T` per indicar la classe o tipus base dels objectes del contenidor.
- `val` serà un paràmetre de tipus `T`.
- `pos` indicarà un iterador que apunta a un element del contenidor i `inici` i `final` seran dos iteradors que indiquen l'interval d'elements [`inici`,`final`).

vector: Constructors

```
vector<T> v;
```

Constructor d'un vector `v` buit.

vector: Constructors

```
vector<T> v;
```

Constructor d'un vector `v` buit.

```
vector<T> v(int n);
```

Constructor d'un vector `v` amb `n` elements, tots ells amb el valor del constructor per defecte de `T`.

vector: Constructors

```
vector<T> v;
```

Constructor d'un vector `v` buit.

```
vector<T> v(int n);
```

Constructor d'un vector `v` amb `n` elements, tots ells amb el valor del constructor per defecte de `T`.

```
vector<T> v(int n, const T& val);
```

Constructor d'un vector `v` amb `n` elements, tots ells amb el valor `val`.

vector: Constructors

```
vector<T> v;
```

Constructor d'un vector `v` buit.

```
vector<T> v(int n);
```

Constructor d'un vector `v` amb `n` elements, tots ells amb el valor del constructor per defecte de `T`.

```
vector<T> v(int n, const T& val);
```

Constructor d'un vector `v` amb `n` elements, tots ells amb el valor `val`.

```
vector<T> v(iterator inici, iterator final);
```

Constructor d'un vector `v` amb els elements de l'interval [`inici`,`final`). Pot ser un interval de qualsevol contenidor de `T`s.

vector: Inserció

```
void vector<T>::push_back(const T& val);
```

Insereix `val` al final. Cost constant amortitzat.

vector: Inserció

```
void vector<T>::push_back(const T& val);
```

Insereix `val` al final. Cost constant amortitzat.

```
iterator vector<T>::insert(iterator pos, const T& val);
```

Insereix `val` a la posició `pos` desplaçant els elements posteriors. Retorna iterador a l'element inserit. Cost lineal.

vector: Inserció

```
void vector<T>::push_back(const T& val);
```

Insereix `val` al final. Cost constant amortitzat.

```
iterator vector<T>::insert(iterator pos, const T& val);
```

Insereix `val` a la posició `pos` desplaçant els elements posteriors. Retorna iterador a l'element inserit. Cost lineal.

```
void vector<T>::resize(int n);
void vector<T>::resize(int n, const T&val);
```

Fa que el vector tingui `n` elements (afegint o eliminant el que calgui). Si hi ha el paràmetre `val` els possibles nous elements tindran aquest valor. Cost lineal.

vector: Inserció

Motiu del cost constant amortitzat del push_back()

- Els elements d'un vector s'han d'emmagatzemar consecutivament a la memòria.
- Cada vegada que un vector ha de créixer, STL reserva més espai del que necessita per aquella operació.
- Mentre hi ha lloc per nous elements el cost és constant.
- Quan queda ple, la següent inserció implica reservar nou espai, copiar-hi tot el vector i posar-hi el nou element (en aquest cas tindrà el cost lineal de moure-ho tot).
- Per més informació, consulteu el mètode `capacity()` .

IMPORTANT: No confongueu `capacity()` amb `size()`, que veurem més endavant.

vector: Esborrat

```
void vector<T>::pop_back();
```

Elimina l'últim element del vector. Cost constant.

vector: Esborrat

```
void vector<T>::pop_back();
```

Elimina l'últim element del vector. Cost constant.

```
iterator vector<T>::erase(iterator pos);
```

Elimina l'element apuntat per pos, desplaçant tots els elements posteriors. Retorna iterador a l'element següent al que s'ha esborrat. Cost lineal.

vector: Esborrat

`void vector<T>::pop_back();`

Elimina l'últim element del vector. Cost constant.

`iterator vector<T>::erase(iterator pos);`

Elimina l'element apuntat per `pos`, desplaçant tots els elements posteriors. Retorna iterador a l'element següent al que s'ha esborrat. Cost lineal.

`void vector<T>::clear();`

Elimina tots els elements del vector. Cost lineal.

vector: Consulta (i)

`T& vector<T>::front();`

Retorna el primer element del vector. Cost constant.

vector: Consulta (i)

`T& vector<T>::front();`

Retorna el primer element del vector. Cost constant.

`T& vector<T>::back();`

Retorna el darrer element del vector. Cost constant.

vector: Consulta (i)

`T& vector<T>::front();`

Retorna el primer element del vector. Cost constant.

`T& vector<T>::back();`

Retorna el darrer element del vector. Cost constant.

`T& vector<T>::operator[](int n);`

Retorna l'element `n`-èssim del vector. No controla rang. Pot usar-se tant per consultes com per modificacions. Cost constant.

vector: Consulta (i)

`T& vector<T>::front();`

Retorna el primer element del vector. Cost constant.

`T& vector<T>::back();`

Retorna el darrer element del vector. Cost constant.

`T& vector<T>::operator[](int n);`

Retorna l'element `n`-èssim del vector. No controla rang. Pot usar-se tant per consultes com per modificacions. Cost constant.

`T& vector<T>::at(int n);`

Retorna l'element `n`-èssim del vector. Si `n` $\geq$ `size()` s'activa una excepció. Cost constant.

vector: Consulta (ii)

`size_type vector<T>::size();`

Retorna quants elements té el vector. Aniran de `0` fins a `size()-1`. Cost constant.

vector: Consulta (ii)

`size_type vector<T>::size();`

Retorna quants elements té el vector. Aniran de `0` fins a `size()-1`. Cost constant.

`bool vector<T>::capacity();`

Retorna quants elements podria tenir el vector sense haver de reassignar la seva posició a memòria. COMPTE: el vector no té `capacity()-1` elements. El vector té `size()-1` elements. Cost constant.

vector: Consulta (ii)

`size_type vector<T>::size();`

Retorna quants elements té el vector. Aniran de `0` fins a `size()-1`. Cost constant.

`bool vector<T>::capacity();`

Retorna quants elements podria tenir el vector sense haver de reassignar la seva posició a memòria. COMPTE: el vector no té `capacity()-1` elements. El vector té `size()-1` elements. Cost constant.

`bool vector<T>::empty();`

Retorna `size() == 0`. Cost constant.

vector: Iteradors (i)

```
iterator vector<T>::begin();
const_iterator vector<T>::begin();
```

Retorna un iterador que apunta al primer element del vector. Cost constant. Si el vector és buit, retorna l'iterador `end()`.

vector: Iteradors (i)

```
iterator vector<T>::begin();
const_iterator vector<T>::begin();
```

Retorna un iterador que apunta al primer element del vector. Cost constant. Si el vector és buit, retorna l'iterador `end()`.

```
iterator vector<T>::end();
const_iterator vector<T>::end();
```

Retorna un iterador que apunta a l'element "després de l'últim". És un error accedir al contingut d'`end()`. Cost constant.

vector: Iteradors (ii)

```
reverse_iterator vector<T>::rbegin();
const_reverse_iterator vector<T>::rbegin();
```

Retorna un iterador invers que apunta a l'últim element del vector (el primer en ordre invers). Cost constant.

vector: Iteradors (ii)

```
reverse_iterator vector<T>::rbegin();
const_reverse_iterator vector<T>::rbegin();
```

Retorna un iterador invers que apunta a l'últim element del vector (el primer en ordre invers). Cost constant.

```
reverse_iterator vector<T>::rend();
const_reverse_iterator vector<T>::rend();
```

Retorna un iterador invers que apunta a l'element anterior al primer (posterior a l'últim en ordre invers). És un error accedir al contingut d'`rend()`. Cost constant.

vector: exemple

Mirem codi...

Vegem codi d'exemple amb el contenidor `vector`.

Fitxer `exempleVector.cpp` (el teniu al Moodle).

Taula de continguts

El contenidor `list`

El contenidor `list` és un contenidor implementat en forma de llista doblement encadenada:

- Permet accés seqüencial als seus elements.
- Permet insercions i esborrats a qualsevol lloc en temps constant.
- Els iteradors que té associats són bidireccionals però no aleatoris.

list: Constructors

```
list<T> l;
```

Constructor d'una list buida.

list: Constructors

```
list<T> l;
```

Constructor d'una list buida.

```
list<T> l(int n);
```

Constructor d'una list amb n elements.

list: Constructors

```
list<T> l;
```

Constructor d'una list buida.

```
list<T> l(int n);
```

Constructor d'una list amb n elements.

```
list<T> l(int n, T& val);
```

Constructor d'una list amb n elements, tots amb valor val.

list: Constructors

```
list<T> l;
```

Constructor d'una list buida.

```
list<T> l(int n);
```

Constructor d'una list amb n elements.

```
list<T> l(int n, T& val);
```

Constructor d'una list amb n elements, tots amb valor val.

```
list<T> l(iterator inici, iterator final);
```

Constructor d'una list amb els elements de l'interval [inici,final). Pot ser un interval de qualsevol contenidor de Ts.

list: Inserció

```
void list<T>::push_front(const T& val);
```

Insereix `val` al principi. Cost constant.

Compte: El `vector` no té aquest mètode

list: Inserció

```
void list<T>::push_front(const T& val);
```

Insereix `val` al principi. Cost constant.

Compte: El `vector` no té aquest mètode

```
void list<T>::push_back(const T& val);
```

Insereix `val` al final. Cost constant.

list: Inserció

```
void list<T>::push_front(const T& val);
```

Insereix `val` al principi. Cost constant.
Compte: El `vector` no té aquest mètode

```
void list<T>::push_back(const T& val);
```

Insereix `val` al final. Cost constant.

```
iterator list<T>::insert(iterator pos, const T& val);
iterator list<T>::insert(iterator pos, iterator inici,
iterator final);
```

La 1a versió insereix `val` a la posició `pos`. Retorna iterador a l'element inserit. Cost constant.
La 2a versió insereix els elements de l'interval `[inici,final)` a partir de `pos`. Cost lineal (número d'elements a inserir).

list: Esborrat

`void list<T>::pop_front();`

Elimina el primer element de la llista. Cost constant.

Compte: El `vector` no té aquest mètode

list: Esborrat

void list<T>::pop_front();

Elimina el primer element de la llista. Cost constant.

Compte: El vector no té aquest mètode

void list<T>::pop_back();

Elimina l'últim element de la llista. Cost constant.

list: Esborrat

`void list<T>::pop_front();`

Elimina el primer element de la llista. Cost constant.
Compte: El `vector` no té aquest mètode

`void list<T>::pop_back();`

Elimina l'últim element de la llista. Cost constant.

`iterator list<T>::erase(iterator pos);`

Elimina l'element apuntat per `pos`. Cost constant.

list: Esborrat

`void list<T>::pop_front();`

Elimina el primer element de la llista. Cost constant.
Compte: El `vector` no té aquest mètode

`void list<T>::pop_back();`

Elimina l'últim element de la llista. Cost constant.

`iterator list<T>::erase(iterator pos);`

Elimina l'element apuntat per `pos`. Cost constant.

`void list<T>::clear();`

Elimina tots els elements de la llista. Cost lineal.

list: Modificació (que no sigui ni inserir ni esborrar)

```
void list<T>::sort();
void list<T>::sort(Compare comp);
```

Ordena la llista. La primera versió aplicant operador $<$ i la segona aplicant la funció `bool comp(const T &t1, const T &t2)`. Cost $n \log(n)$.

Compte: El `vector` no té aquest mètode

list: Consulta

T& list<T>::front();

Retorna el primer element de la llista. Cost constant.

list: Consulta

T& list<T>::front();

Retorna el primer element de la llista. Cost constant.

T& list<T>::back();

Retorna el darrer element de la llista. Cost constant.

list: Consulta

T& list<T>::front();

Retorna el primer element de la llista. Cost constant.

T& list<T>::back();

Retorna el darrer element de la llista. Cost constant.

COMPTE

A diferència del vector les list no tenen accés directe: no existeix ni l'operador [](int n) ni el mètode at(int n).

list: Consulta

T& list<T>::front();

Retorna el primer element de la llista. Cost constant.

T& list<T>::back();

Retorna el darrer element de la llista. Cost constant.

size_type list<T>::size();

Retorna quants elements té la llista. Cost constant.

list: Consulta

`T& list<T>::front();`

Retorna el primer element de la llista. Cost constant.

`T& list<T>::back();`

Retorna el darrer element de la llista. Cost constant.

`size_type list<T>::size();`

Retorna quants elements té la llista. Cost constant.

`bool list<T>::empty();`

Retorna `size() == 0`. Cost constant.

list: Iteradors (i)

```
iterator list<T>::begin();
const_iterator list<T>::begin();
```

Retorna un iterador que apunta al primer element de la llista. Cost constant. Si la llista és buida, retorna l'iterador `end()`.

list: Iteradors (i)

```
iterator list<T>::begin();
const_iterator list<T>::begin();
```

Retorna un iterador que apunta al primer element de la llista. Cost constant. Si la llista és buida, retorna l'iterador `end()`.

```
iterator list<T>::end();
const_iterator list<T>::end();
```

Retorna un iterador que apunta a l'element "després de l'últim". És un error accedir al contingut d'`end()`. Cost constant.

list: Iteradors (ii)

```
reverse_iterator list<T>::rbegin();
const_reverse_iterator list<T>::rbegin();
```

Retorna un iterador invers que apunta a l'últim element de la llista (el primer en ordre invers). Cost constant.

list: Iteradors (ii)

```
reverse_iterator list<T>::rbegin();
const_reverse_iterator list<T>::rbegin();
```

Retorna un iterador invers que apunta a l'últim element de la llista (el primer en ordre invers). Cost constant.

```
reverse_iterator list<T>::rend();
const_reverse_iterator list<T>::rend();
```

Retorna un iterador invers que apunta a l'element anterior al primer (posterior a l'últim en ordre invers). És un error accedir al contingut d'`rend()`. Cost constant.

list: exemple

Mirem codi...

Vegem codi d'exemple amb el contenidor `list`.

Fitxer `exempleList.cpp` (el teniu al Moodle).

Taula de continguts

Feina a fer

Exercici 1 de laboratori

El teniu en un PDF separat disponible al Moodle.

jordi.regincos@udg.edu