E.D.A. – Sessió 1

STL (**S**tandard **T**emplate **L**ibrary) (i)

Vector i **Iteradors**

Jocs de Proves

Universitat de Girona
Escola Politècnica Superior

Professor (grups 1, 4 i 5)

Miquel Bofill Arasa

- Dept. Informàtica, Matemàtica Aplicada i Estadística
- Despatx 254 edifici P4
- c/e miquel.bofill@udg.edu

Tutories i/o resolució de dubtes:

- Fòrum “Fòrum de laboratori” del Moodle
- Presencials: hores convingudes
- Correu electrònic (assumpte començat per “[EDA]”)

Professor (grups 2 i 3)

Jordi Regincós Isern

- Dept. Informàtica, Matemàtica Aplicada i Estadística
- Despatx 234 edifici P4
- c/e jordi.regincos@udg.edu

Tutories i/o resolució de dubtes:

- Fòrum “Fòrum de laboratori” del Moodle
- Presencials: Dimarts d'onze a una i de tres a cinc
- Correu electrònic (assumpte començat per “[EDA]”)

Índex

Introducció STL

Standard **T**emplate **L**ibrary és una biblioteca de C++ que conté:

- **Contenidors genèrics** (vector, deque, list, set, map,…)
- **Iteradors** sobre els contenidors (directes, inversos, aleatoris,…)
- **Algoritmes genèrics** per aplicar als contenidors (find, sort, merge,…)
- Altres elements (adaptadors, objectes-funció,…)
- **Tots els elements poden acloblar-se entre ells**

Introducció STL / Contenidors

Els principals contenidors són:

- **Vector**: un vector que creix dinàmicament
- **List**: una llista amb inserció i esborrat eficients
- **Deque**: una cua amb possibilitat de posar i treure a tots dos extrems
- **Map**: una taula de parelles <clau,valor> sense claus repetides
- **Set**: un conjunt sense elements repetits
- **Multimap, Multiset**: equivalents al map i al set, però amb claus o elements repetits, respectivament

Introducció STL / Iteradors (i)

Permeten realitzar recorreguts sobre els elements dels contenidors.

Tipologia d'iteradors en la STL:

- **Input:** permet consultar els elements, cap endavant.
- **Output:** permet modificar els elements, cap endavant.
- **Forward:** input+output, cap endavant.
- **Bidirectional:** input+output, endavant i endarrera.
- **Random:** permet accés als elements en qualsevol ordre (no necessàriament seqüencial).

Introducció STL / Iteradors (ii)

- No tots els contenidors permeten tots els iteradors.
- Operadors permesos en els iteradors:
  - *Input*, *Output*: **==**, **!=**, ***** (accés a l'element apuntat), **++**
  - *Forward*: Els de *Input* i *Output*
  - *Bidirectional*: Els de *Forward* i **--**
    - *Random*: Els de *Bidirectional* i **+=**, **-=**, **+**, **-**, **<**, **>**, **>=**, **<=**, **[]**
- Els algoritmes genèrics treballen sobre els iteradors

Introducció STL / Recursos (i)

http://en.cppreference.com/

cppreference.com

cplusplus.com - The C++ Resources Network

http://www.cplusplus.com/

Webs que cobreixen diversos aspectes del C++

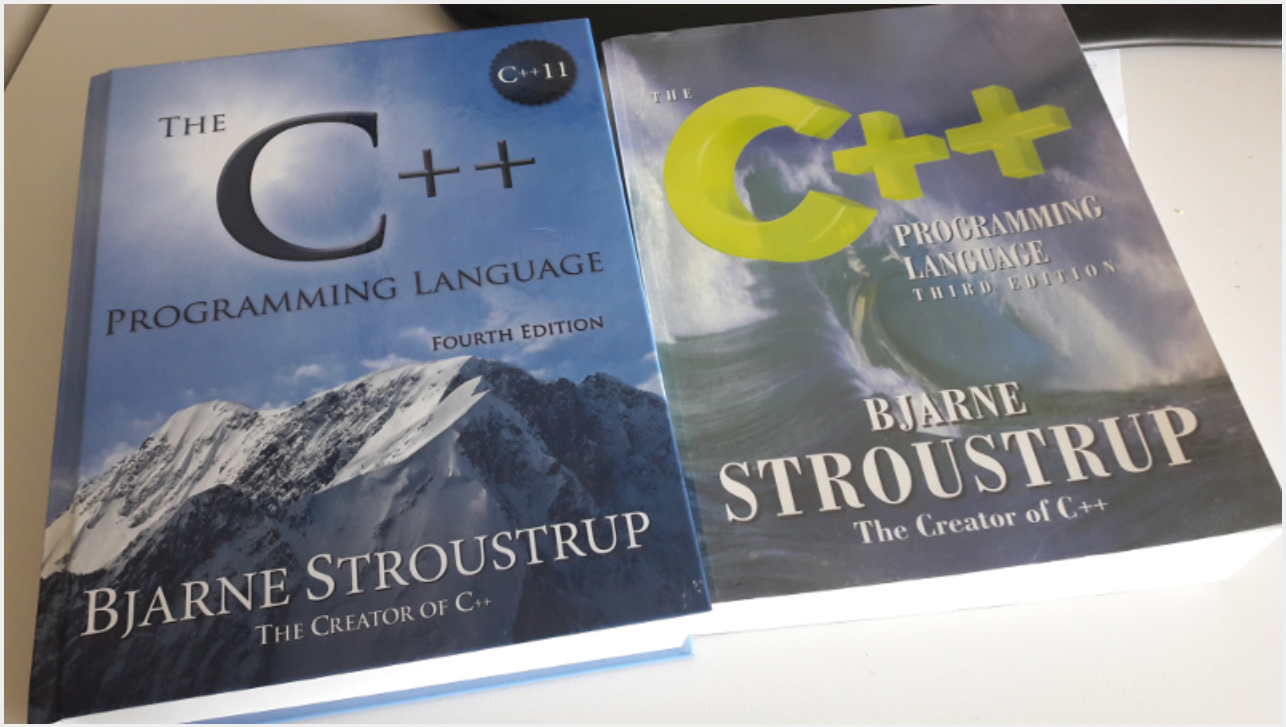
Introducció STL / Recursos (ii)

La referència fonamental

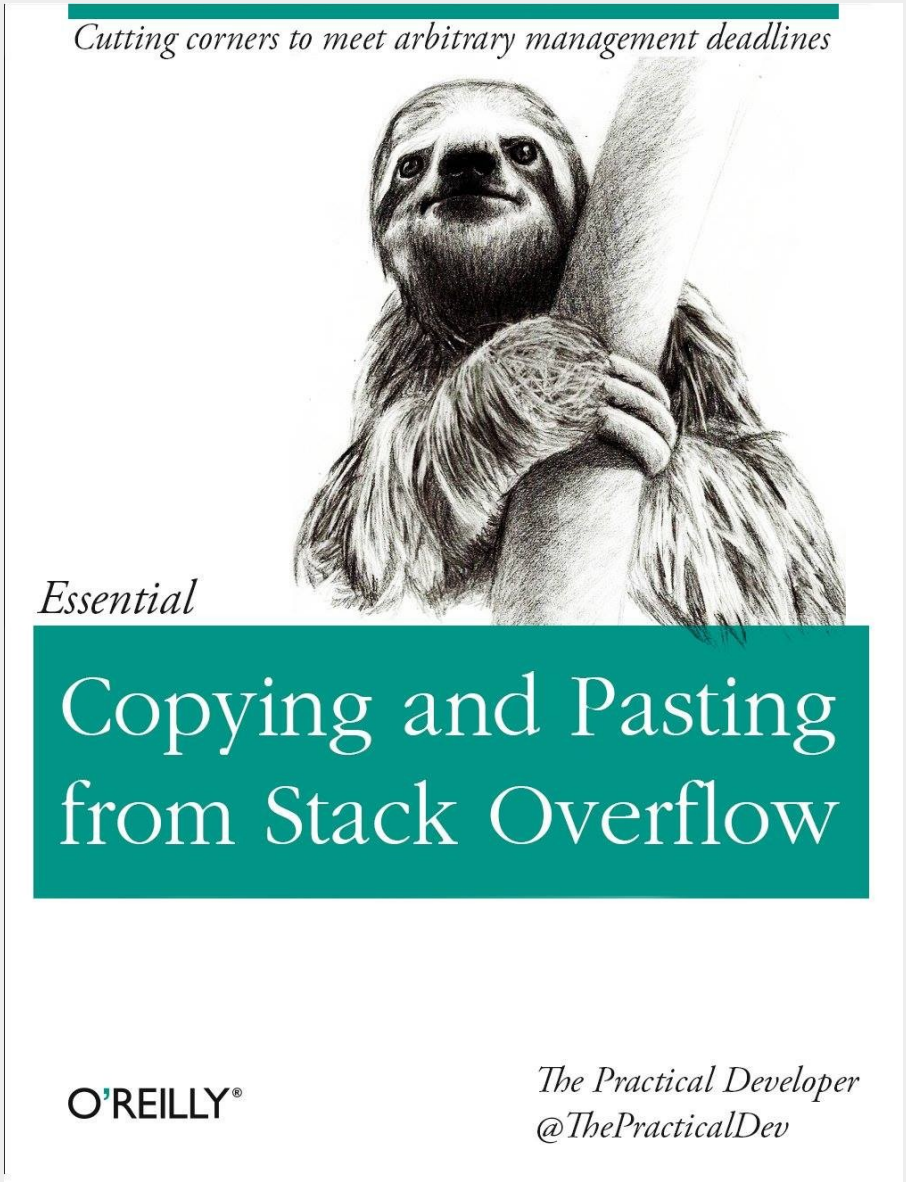
The C++ Programming Language

Bjarne Stroustrup

(Addison-Wesley)



NO recomanem...



Vectors STL

Un vector STL és un contenidor genèric

- Permet accés aleatori als seus elements en temps constant
- Permet insercions i esborrats al final en temps (amortitzat) constant
- Permet insercions i esborrats al mig en temps lineal
- No té limitacions de mida: el vector “s’expandeix” automàticament si cal guardar més elements
- Té associats iteradors aleatoris (Random Access), que permeten ++, --, +=, -=, ...

http://en.cppreference.com/w/cpp/container/vector

Vector STL / Constructors

vector<T> v;

Constructor per defecte d'un vector **v** buit

vector<T> v(int n);

Constructor d'un vector **v** amb **n** elements, tots ells amb el contingut del constructor per defecte de la classe **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 first, iterator last)

Constructor d'un vector v amb tots els elements entre els dos iteradors

Vector STL / Inserció

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

Insereix el valor **val** al final del vector, amb cost amortitzat constant. Incrementa la mida del vector.

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

Insereix **val** a la posició **pos**, desplaçant els elements posteriors. Té cost lineal. Incrementa la mida del vector. Retorna un iterador que apunta a l'element inserit.

Vector STL / Esborrat

void vector<T>::pop_back();

Elimina el darrer element del vector, amb cost amortitzat constant.

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

Elimina l'element de la posició **pos**, desplaçant els elements posteriors. Té cost lineal.

void vector<T>::clear();

Elimina tots els elements del vector.

Vector STL / Consultes (i)

iterator vector<T>::begin();

Retorna un iterador aleatori que referencia al primer element.

iterator vector<T>::end();

Retorna un iterador aleatori que referencia a l'element teòric que hi hauria després del darrer element (no es pot desreferenciar).

reverse_iterator vector<T>::rbegin();

Retorna un **iterador invers** aleatori que referencia al darrer element: el d'abans del referenciat per **end()**

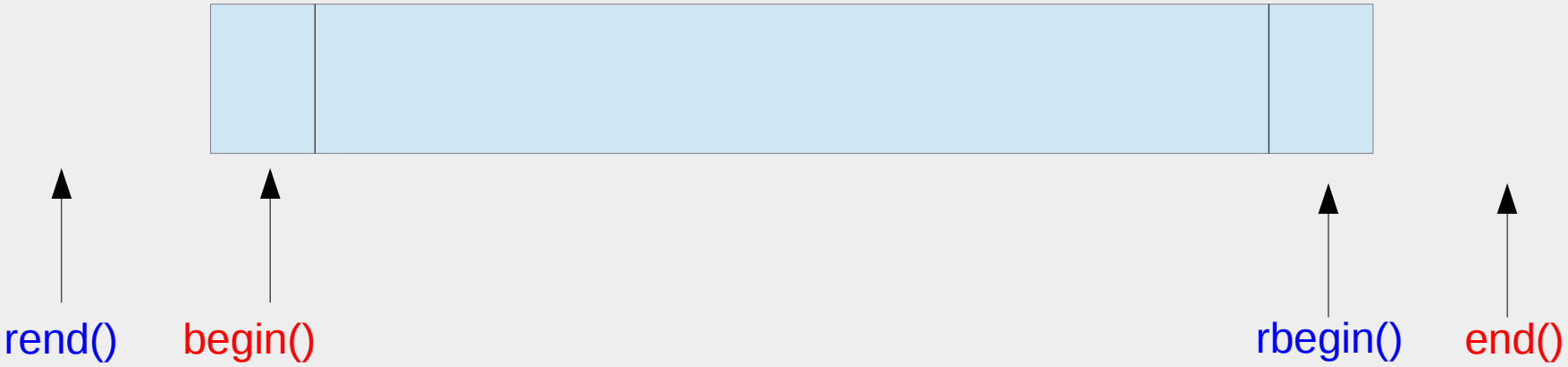
reverse_iterator vector<T>::rend();

Retorna un **iterador invers** aleatori que referencia a l'element teòric que precedeix el primer element: el d'abans del referenciat per **begin()**.

Nota: en els iteradors inversos, l'operador ++ “avança” cap al principi de l'estructura.

Vector STL / Consultes (ii)

Posició dels iteradors



Vector STL / Consultes (iii)

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

Retorna el darrer element del vector, sense esborrar-lo.

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

Retorna el primer element del vector, sense esborrar-lo.

size_type vector<T>::size();

Retorna el nombre d'elements del vector.

bool vector<T>::empty();

Diu si el vector està buit.

size_type vector<T>::capacity();

Retorna nombre màxim d'elements que es poden afegir en el vector sense que s'expandeixi. Quan s'expandeix dobla la mida. **size_type** és un enter sense signe.

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

Retorna el valor de la posició indicada. La posició ha d'existir. En retornar una referència, pot usar-se tant per consultar com per assignar valors nous al vector.

void vector<T>::resize(int n);

Ajusta el vector de manera que tingui exactament **n** elements.

Exemple: operador [ ]

```
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector (10);   // 10 zero-initialized elements

  std::vector<int>::size_type sz = myvector.size();

  // assign some values:
  for (unsigned i = 0; i < sz; i++) myvector[i] = i;

  // reverse vector using operator[]:
  for (unsigned i = 0; i < sz/2; i++)
  {
    int temp;
    temp = myvector[sz-1-i];
    myvector[sz-1-i] = myvector[i];
    myvector[i] = temp;
  }

  std::cout << "myvector contains:";
  for (unsigned i = 0; i < sz; i++)
    std::cout << ' ' << myvector[i];
  std::cout << '\n';

  return 0;
}

Output:
myvector contains: 9 8 7 6 5 4 3 2 1 0
```

Jocs de proves

Tots els progames que lliureu a EDA hauran d'anar acompanyats d'un joc de proves (un o més fitxers de text amb diferents seqüències de dades d'entrada).

Us deixarem a l'apartat de pràctiques d'EDA al Moodle un document on s'explica com fer-los.

No incloure el joc de proves **pot implicar no aprovar** l'activitat de laboratori o la pràctica.

Exercici

Idea: implementar una classe `Comarca` i una classe `Pais` (que contindrà un vector de `Comarca`) més un petit `main.cpp` per provar-ho fent el que es demana a l'enunciat.

`Comarca`:

```
int _codi;
string _nom;
int _poblacio;
float _extensio;
```

`Pais`:

```
int _numComarques;
int _poblacio;
float _extensio;
vector <Comarca> _comarques;
```

Teniu els detalls al

Moodle