



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Standard Template Library (STL) **vector, list, string**

Stefano Ghidoni



Agenda

- Tre container a confronto
- Funzioni che invalidano gli iteratori: `insert()` ed `erase()` su `vector`
- Overview dei contenitori STL



Confronto tra contenitori

- Abbiamo già usato tre contenitori STL:
 - std::vector
 - std::string
 - Confronto con char[]
 - std::list
- Analizziamo le caratteristiche principali



- std::vector
 - Supporta tutte le operazioni, inclusi insert() e erase()
 - Fornisce []
 - Insert() ed erase() sono inefficienti (devono muovere molti elementi in memoria)
 - Un problema per collezioni di dati molto grandi
 - Fornisce range check (su richiesta!)
 - Espandibile
 - Elementi contigui in memoria
 - Operatore == confronta gli elementi

Differenza con gli
array stile C



- std::string
 - Come i vettori, ma aggiungono le operazioni di manipolazione del testo
 - Concatenazione: +, +=
 - Operatore == confronta gli elementi



string vs char[]

- Si noti la differenza tra string e char[]
- char[] è un array stile C, quindi:
 - Non conosce la propria dimensione
 - Non possiede begin(), end() né altri iteratori
 - Non possiede range check
 - Elementi contigui in memoria
 - Dimensione fissa, decisa a tempo di compilazione
 - Confronto (==, !=) e output (<<) si riferiscono al puntatore al primo elemento, non al contenuto
 - << riesce a gestire il puntatore



- std::list
 - Fornisce le operazioni abituali, ma non subscripting – []
 - Possiamo usare insert() e erase() senza spostare gli altri elementi
 - Espandibile
 - Operatore == confronta gli elementi



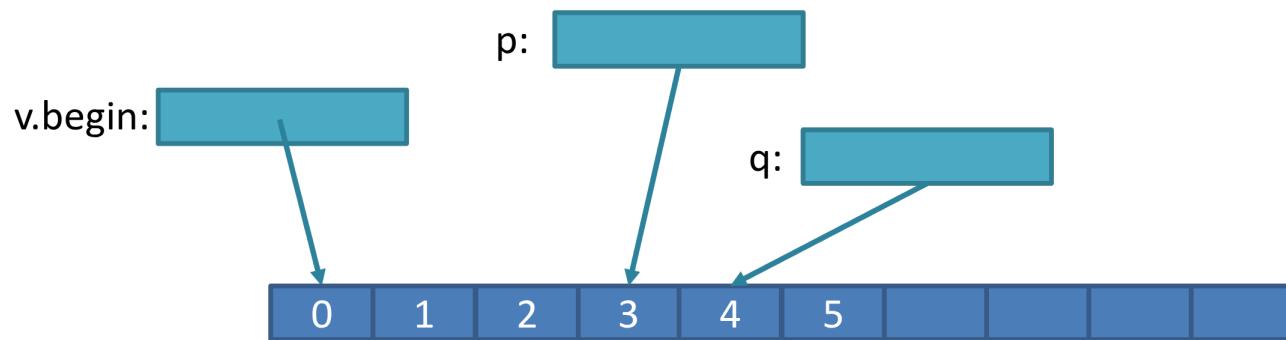
insert() e erase() con vector

- Effettuare una insert() o una erase() in uno std::vector può essere un'operazione inefficiente
- Muovere gli elementi ha una forte implicazione: **rende non validi gli iteratori** che stanno puntando allo std::vector



insert() e erase() con vector

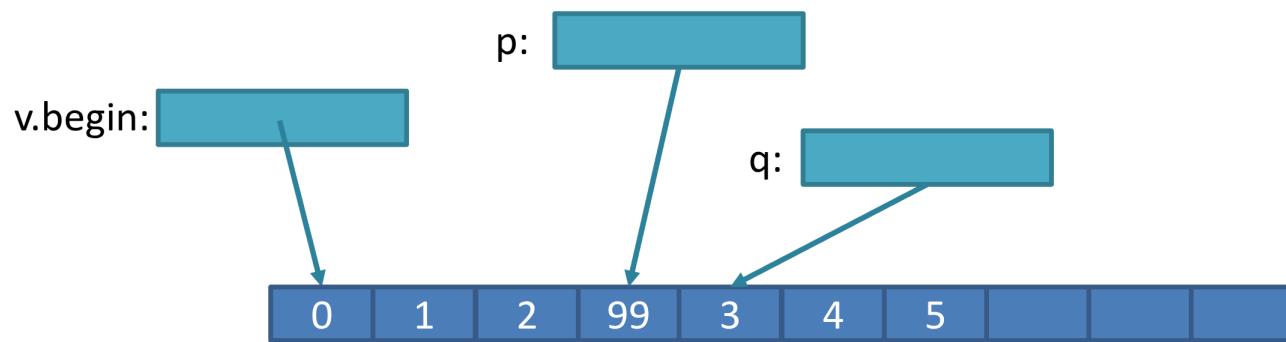
```
std::vector<int>::iterator p = v.begin();
++p; ++p; ++p;
auto q = p;
++q;
```





insert() e erase() con vector

```
p = v.insert(p, 99);
```





insert() ed erase() con le liste

- Si ripropone lo stesso problema con le liste?
Perché?

No: inserimento e rimozione di un oggetto su una lista modificano il collegamento tra gli elementi, quindi il problema non si verifica.



Overview dei contenitori STL

- STL fornisce molti contenitori
 - vector
 - list
 - deque
 - map (albero bilanciato e ordinato)
 - multimap
 - unordered_map
 - unordered_multimap
 - set
 - multiset
 - unordered_set
 - unordered_multiset
 - array



Contenitore STL

- Cos'è un contenitore STL?
- La definizione è complessa, però un contenitore:
 - È una sequenza di elementi [begin():end())]
 - Supporta le operazioni di copia (usate nell'assegnamento o nel costruttore di copia)
 - Chiama il tipo dei propri elementi `value_type`



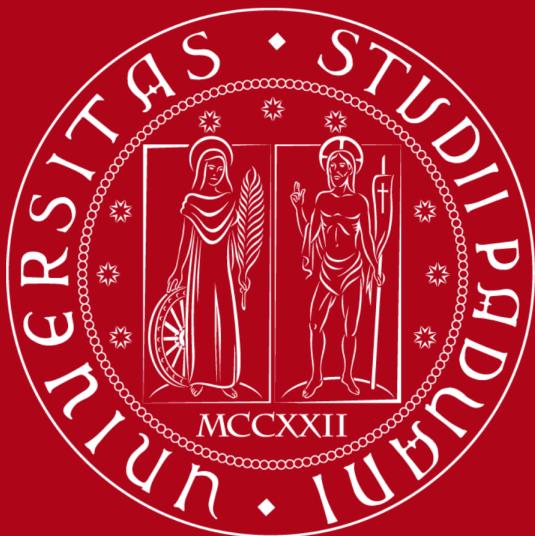
Contenitore STL

- Possiede tipi iteratore chiamati iterator e const_iterator
 - Gli iteratori forniscono *, ++ (pre e post), == e !=
 - Gli iteratori di vector forniscono anche --, [], + e - (random access iterators)
- Fornisce insert() ed erase(), front() e back(), push_back() e pop_back(), size(), ...
 - vector e map forniscono anche []
- Forniscono operatori di confronto: ==, !=, <, <=, >, >=



Recap

- Tre contenitori a confronto:
 - std::vector,
 - std::string,
 - std::list
- insert() ed erase() su std::vector – effetto sugli iteratori
- Overview dei contenitori STL
- Elementi comuni dei contenitori STL



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Standard Template Library (STL) **vector, list, string**

Stefano Ghidoni