34. Container ed iteratori

Corso di Informatica

Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
23/12/2020

Outline

- Container
  - Sequence container
  - Associative container
  - Unordered associative container
- Iteratori

Container

- I **container** sono uno tra i costrutti maggiormente utilizzati nel C++
- Sono definiti nella Standard Template Library
- Ne esistono per molte delle strutture dati che abbiamo visto ed utilizzato finora
- Tre raggruppamenti:
  - **Sequence container**
  - **Associative container**
  - **Unordered associative container**

Container – Sequence container

- Modellano i dati al loro interno sotto forma di **sequenza**
  - *Assicurano quindi un accesso di tipo sequenziale*
- La STL offre:
  - `array`: *container per array non ridimensionabile;*
  - `vector`: *container per un vettore che può essere ridimensionato a runtime;*
  - `deque`: *container per una coda doppia (non abbiamo visto questa struttura);*
  - `list`: *container per una **doubly linked list**;*
  - `forward_list`: *container per una **linked list***

Container – Associative container

- Permettono di usare degli **array associativi**
- Sono particolari strutture che memorizzano delle **chiavi** e/o dei **valori**
- La STL offre:
  - `set`: *container per un insieme di chiavi ordinate ed univoche;*
  - `map`: *container per un insieme di coppie chiave – valore univoci;*
  - `multiset`: *container per un insieme di chiavi, anche ripetute;*
  - `multimap`: *container per un insieme di coppie chiave – valore, anche ripetute.*
- Gli elementi contenuti in questi container sono **ordinati**, ovvero vale il principio dello **strict weak ordering**

Container – Unordered Associative container

- **Weak ordering**

$$\forall (a, b), se\ a \leq b, allora\ non\ sussiste\ b \leq a$$
$$se\ a \leq b\ e\ b \leq a, allora\ a \sim b$$
$$se\ a \leq b\ e\ b \leq c, allora\ a \leq c$$
$$se\ a \sim b\ e\ b \sim c, allora\ a \sim c$$

- **Strict Weak ordering**

$$se\ a \leq b\ e\ a\ !\sim b, allora\ a < b$$

- I container associativi **non** ordinati non rispettano lo strict weak ordering, quindi gli elementi al loro interno non sono ordinati
- Abbiamo quindi **`hash_map`**, **`hash_set`**, **`hash_multiset`** ed **`hash_multimap`**

Container – Ordinati o non ordinati?

- I container associativi ordinati sono sostanzialmente analoghi alle loro controparti non ordinate
- La differenza sta nel fatto che i container non ordinati memorizzano una versione ‘compressa’ dell’oggetto (**hash**)
  - *N.B. il concetto di hash è molto più esteso, e formalmente differente da quello di compressione. Un hash è il risultato dell’applicazione di una funzione non invertibile alla codifica in bit dell’oggetto originario; trattarlo è fuori dagli scopi di questo corso*
- La scelta quindi si traduce tra l’utilizzo di una versione ‘compatta’ (e quindi più veloce) del container, ed una che offre maggiori funzionalità (soprattutto in termini di accesso ordinato ai dati, come nel caso di grafi)

Container – Vantaggi nell'utilizzo

- Un vantaggio immediato derivante dall'utilizzo di un container sta nella possibilità di utilizzare una serie di funzioni standard definite su ognuno di esso
- Ad esempio, possiamo svincolare l'accesso agli elementi di un array dalla conoscenza della dimensione dello stesso usando la funzione **`size`**:

```
for (int i = 0; i < 10; i++)
{
    cout << vettore[i] << endl;
}
```

```
for (int i = 0; i < v.size(); i++)
{
    cout << vettore[i] << endl;
}
```

- Il vantaggio principale però sta nell'uso degli **iteratori**

Iteratori

- Un **iteratore** è un costrutto che permette di ‘scorrere’ gli elementi di un container
- L’iteratore parte da un elemento del container, termina in un altro, e restituisce, per ogni elemento, un puntatore allo stesso
- La sintassi è leggermente differente:

```
for (vector<int>::iterator it = vettore.begin(); it != vettore.end(); it++)
{
    cout << *it << endl;
}
```

- In questo esempio, definiamo un iteratore facendo in modo che punti all’inizio del container (**`begin()`**) e che scorra sino al termine dello stesso (**`end()`**)
- Notiamo come l’iteratore **punti** al singolo elemento, il quale risulta accessibile mediante dereferenziazione

Iteratori

- Alcune note:
  - *Le funzioni degli iteratori sono le stesse per i diversi tipi di container. Questo ci permette di adattare il nostro codice a container differenti in modo molto semplice!*
  - *Accedere ai membri di un container come una* **`list`** *usando un classico* **`for`** *ed il metodo* **`size()`** *per ricavarne le dimensioni comporta due criticità: la prima è che le liste non supportano l'indicizzazione mediante l'operatore* **`[]`**, *la seconda è che il metodo* **`size()`** *dovrà fare una scansione preliminare della lista per conoscere la dimensione, che invece è già nota in un array, comportando una perdita in termini di performance. Questo ovviamente non accade se si utilizza un iteratore.*

Domande?

42