Standard Template Library

Alberto Ferrari

Standard Template Library

- Progettata per gestire insiemi di dati in modo comodo ed efficiente senza conoscere dettagli implementativi
- Fa parte dello standard C++
- È basata sulla programmazione generica

STL

- **Containers Class (contenitori)**
  - Sono classi template usate per contenere insiemi di dati
  - Si distinguono in tre categorie:
    - Sequenziali
    - Associativi (permettono l'accesso con chiave)
    - Adattatori
- **Iterators (iteratori)**
  - Servono per muoversi nell'insieme di dati
- **Algorithms (algoritmi)**
  - Implementazioni di molti algoritmi per operare sull'insieme di dati

Alberto Ferrari

Contenitori sequenziali

- Gli elementi sono ordinati in base al modo in cui sono inseriti
- **`slist`** (lista concatenata semplice, non standard)
- **`list`** (lista concatenata doppia)
- **`vector`** (accesso mediante indice)
- **`deque`** (coda in cui si inseriscono e rimuovono elementi a entrambe le estremità)

Alberto Ferrari

Contenitori associativi

- Ogni elemento è composto da **chiave** e **valore**
- Gli elementi sono individuati per mezzo della chiave
- **`set, multiset, map, multimap`**

Alberto Ferrari

Adattatori

- Sono classi template implementate a partire da altre classi
  - **stack**, **queue** basate su **deque**
    - **stack (LIFO), queu (FIFO)**
  - **priority_queue** basata su **vector**
- È possibile specificare un container sottostante differente da quello di default
  - Es: **stack<int, vector<int> >**

Alberto Ferrari

Esempio di contenitore sequenziale: vector

- I vector sono array la cui dimensione può cambiare durante l'esecuzione
- La classe è definita nella libreria **`vector`** all'interno del namespace **`std`**

```
#include <vector>
using namespace std;
```

Alberto Ferrari

Accesso agli Elementi

- La notazione []
  - può essere usata per
    - leggere un elemento
    - cambiare un elemento che ha già un valore
  - non può essere usata per
    - inizializzare un elemento
  - non controlla se l'elemento esiste
- Per aggiungere elementi (in ordine di posizione) si usa il metodo **`push_back()`**

Alberto Ferrari

Costruttore con un Argomento Intero

- Crea un numero di elementi pari all'argomento e li inizializza
  - Con lo zero del tipo numerico se è un vettore di numeri
  - Con il costruttore di default se il tipo base è un tipo classe

Esempio:

```
vector<int> v(10);
vector<Tempo> ore(12);
```

Alberto Ferrari

Dimensioni e Capacità

- La **dimensione** è il numero di elementi in un vector (restituita dal metodo **size()**)
- La **capacità** è il numero di elementi per i quali c'è memoria allocata
- Se aggiungo elementi a un vector che ha esaurito la capacità, questa viene automaticamente aumentata

Alberto Ferrari

Metodi reserve() e resize()

- E’ possibile ignorare la gestione della capacità se non serve efficienza
- Altrimenti posso usare il metodo **reserve()** per riservare una capacità minima
- Il metodo **resize()** modifica la dimensione del vector

esVector.cpp

Alberto Ferrari

Iteratori

- Consentono di accedere agli elementi di un contenitore indipendentemente dalla sua struttura
- Sono progettati per nascondere i dettagli implementativi e fornire un modo uniforme per esplorare le strutture dati
- Ogni classe container ha il suo tipo di iteratori ma la terminologia, la sintassi e la semantica sono le stesse
- Un iteratore astrae il concetto di puntatore (formalmente **non** è un puntatore)

```
TipoContenitore<T>::iterator nomeIteratore
```

Alberto Ferrari

Operatori sugli Iteratori

- **Operatore di incremento ++** (in forma prefissa e postfissa)
  - fa avanzare l'iteratore all'elemento successivo
- **Operatore di decremento --** (in forma prefissa e postfissa)
  - Fa retrocedere l'iteratore all'elemento precedente
- **Operatori di confronto == e !=**
  - Per verificare se due iteratori puntano allo stesso elemento
- **Operatore di dereferenziazione ***
  - Per accedere all'elemento puntato dall'iteratore (l'accesso può essere di sola lettura)

Alberto Ferrari

Iteratori e Containers

- Le classi container hanno due funzioni membro che restituiscono iteratori collocati in posizioni speciali
  - **`begin()`** restituisce un iteratore che punta al primo elemento
  - **`end()`** restituisce un valore speciale che può essere usato per verificare se un iteratore punta oltre l'ultimo elemento

Alberto Ferrari

Tipi di Iteratori

- Gli iteratori sono classificati secondo il tipo di operazioni ad essi applicabili
  - **Forward iterators**: si può applicare l'operazione ++
  - **Bidirectional iterators**: si possono applicare le operazioni ++ e --
  - **Random access iterators**: si possono applicare le operazioni ++ e -- e si può accedere a qualsiasi elemento in un solo passo
- Ogni categoria include le precedenti

`esVectorIterator.cpp`

Alberto Ferrari

Iteratori constant e mutable

- Un iteratore di qualsiasi tipo è
  - **Constant** se l'operatore * restituisce l'elemento puntato come r-value
  - **Mutable** se l'operatore * restituisce l'elemento puntato come l-value
- Se una classe container ha iteratori mutable, ha anche iteratori const

Alberto Ferrari

Reverse Iterators

- Se una classe container ha iteratori bidirezionali, per passare gli elementi in ordine inverso si possono usare i reverse iterators
- La funzione membro **rbegin()** restituisce un iteratore che punta all'ultimo elemento
- La funzione membro **rend()** restituisce un valore speciale che può essere usato per verificare se un reverse iterator punta oltre il primo elemento
- L'operatore ++ fa avanzare un reverse iterator in senso inverso

esVectorReverseIterator.cpp

Alberto Ferrari

Esempio di contenitore sequenziale: list

- Container sequenziale ottimizzato per frequenti richieste di inserimento ed eliminazione di elementi
- Doubly linked list: lista bidirezionale percorribile nei due sensi
- Gli elementi non sono contigui in memoria
- Non forniscono accesso casuale agli elementi
- L'inserimento e la rimozione degli elementi impiegano lo stesso tempo in tutte le posizioni

esList.cpp

Alberto Ferrari

Esempio di contenitore associativo: set

- Ogni elemento coincide con la sua chiave
- Ogni elemento può comparire al più una volta
- Per motivi di efficienza memorizza gli elementi in ordine rispetto al loro valore (i set sono implementati come alberi binari)

Alberto Ferrari

Set (ordinamento)

- Per default l'ordinamento usa l'operatore <
- È possibile specificare un diverso criterio di ordinamento purché sia
  - **Antisimmetrico**: op(x,y)=TRUE -> op(y,x)=FALSE
  - **Transitivo**: op(x,y)=TRUE && op(y,z)=TRUE -> op(x,z) = TRUE
  - **Irriflessivo**: op(x,x) = FALSE

Alberto Ferrari

Esempio di contenitore associativo: map

- È un insieme di coppie ordinate di elementi (pair) formate da chiave (first) e dato (second)
  - Es: `map<string, int> voto;`
- Per motivi di efficienza memorizza gli elementi in ordine rispetto al valore della chiave
- Se non si specifica un ordinamento, viene usato quello di default

esMap.cpp

Multiset e Multimap

- **`multiset`** è come **`set`** ma consente la ripetizione degli elementi
- **`multimap`** è come **`map`** ma consente che più valori siano associati alla stessa chiave

Generic Algorithms

- STL contiene algoritmi generici indipendenti dal contenitore
- STL specifica non solo
  - sintassi (interfaccia)
  - semantica (relazioni di I/O)

  ma anche
  - tempo di calcolo

esAlgoritmi.cpp

Alberto Ferrari