Classe list STL: Iteratori e differenze con la classe vector STL

Corso di Programmazione ad Oggetti
Corso di Laurea in Informatica
Università della Calabria
https://www.mat.unical.it/informatica/ProgrammaAdOggetti

STL `list`

Occorre: **`#include <list>`** (e **`using namespace std`**)

È una classe *template*:

- **`list<int> integer_list;`**
- **`list<string> string_list;`**
- **`list<Contatto> contatti_list;`**

Implementazione versatile ed ottimizzata

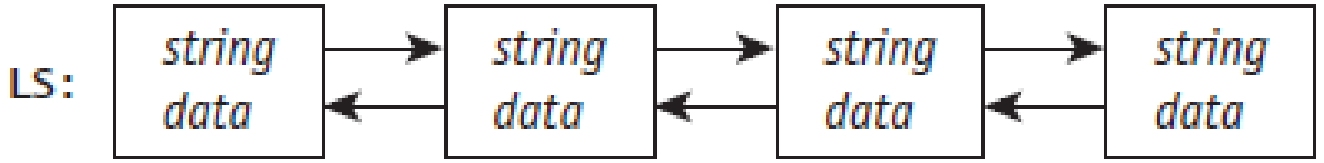
STL `list`

Implementata come **doppiamente concatenata**

- Ogni nodo è collegato al precedente e al successivo
- Iterazione bidirezionale

Esempio:

```
list<string> LS;
```



Iteratori

Gli iteratori sono uno strumento per scandire una struttura dati STL un elemento per volta

Esistono diverse tipologie di iteratori riadattate per consentire di scandire in modo appropriato le diverse tipologie di strutture dati

Riprendiamo il nostro esempio:

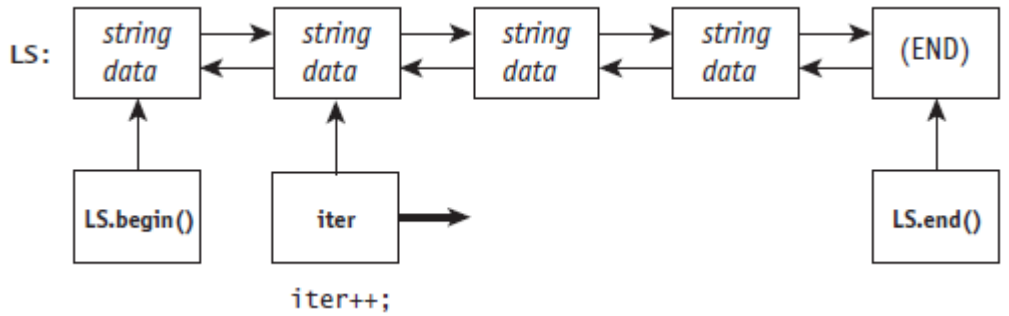
```
// Dichiarazione di un iteratore su una lista di stringhe
list<string>::iterator iter;
// Inizializzazione iteratore sul primo elemento della lista
iter = LS.begin();
// Dereferenziazione di un iteratore:
string s = *iter;
```

Si utilizzano in modo simile ai puntatori, ma NON sono puntatori!

Iteratori e scansione di una lista

Meccanismo per la scansione di una `list`: ***Iteratori bidirezionali***

```
for(list<string>::iterator iter=LS.begin();iter!=LS.end();iter++){
      cout<<(*iter)<<endl;
}
```



`const_iterator` per impedire modifiche agli elementi o iterare su lista non modificabile:

```
for(list<string>::const_iterator iter=LS.begin();iter!=LS.end();iter++){
      cout<<(*iter)<<endl;
}
```

Principali metodi in STL `list`

Consultare la documentazione: http://www.cplusplus.com/reference/list/list/

Metodo	Descrizione
begin	Restituisce un iteratore che «punta» al primo nodo.
end	Restituisce un iteratore che «punta» ad un nodo *FITTIZIO* finale.
size	Restituisce la dimensione, ovvero il numero di elementi.
empty	Restituisce *true* se la lista è vuota, *false* altrimenti.
front	Restituisce il primo elemento se esiste, altrimenti si ottiene un «undefined behavior».
back	Restituisce l'ultimo elemento se esiste, altrimenti si ottiene un «undefined behavior».
push_back	Inserisce un elemento alla fine.
push_front	Inserisce un elemento all'inizio.
insert	Dato un iteratore che «punta» ad un nodo in una certa posizione ed un elemento, in inserire tale elemento prima della posizione specificata.
pop_back	Rimuove l'ultimo elemento se esiste, altrimenti si ottiene un «undefined behavior».
pop_front	Rimuove il primo elemento se esiste, altrimenti si ottiene un «undefined behavior».
erase	Dato un iteratore che «punta» ad un nodo elimina l'elemento corrispondente.
remove	Dato un elemento, lo cerca e rimuove tutte le sue occorrenze (Usa l'operatore ==).
clear	Rimuove tutti gli elementi, rendendo la lista vuota, ovvero con dimensione pari a zero.
unique	Rimuove gli elementi duplicati *CONSECUTIVI* (Usa l'operatore ==).
sort	Ordina la lista (Usa l'operatore <).

STL `vector`

Occorre: **`#include <vector>`** (e **`using namespace std`**)

È una classe *template*:

- **`vector<int> integer_vector;`**
- **`vector<string> string_vector;`**
- **`vector<Contatto> contatti_vector;`**

Locazioni di memoria contigue

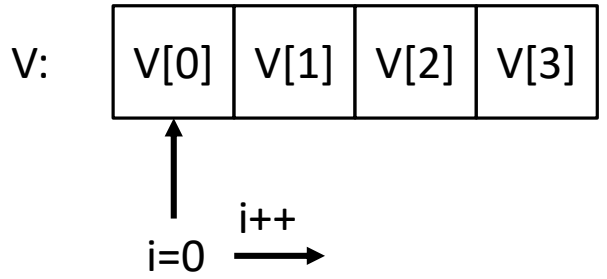
- Accesso casuale (diretto, «random»)
- Gestione automatica della memoria dinamica allocata
  - Quando non c'è sufficiente memoria a disposizione, viene allocata un'area di memoria contigua più grande, ricopiati gli elementi originali e deallocata la vecchia memoria

Iteratori e scansione di un vettore

Meccanismo classico:

```
for(unsigned int i=0;i<V.size();i++){
      cout<<V[i]<<endl;

}
```

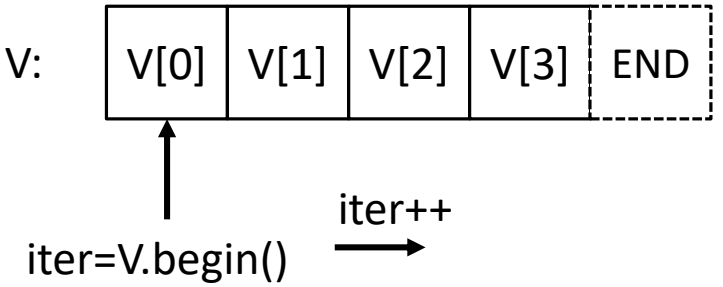


Ulteriore meccanismo per la scansione di un vettore attraverso ***Iteratori ad accesso diretto***

```
vector<string> V;

for(vector<string>::iterator iter=V.begin();
    iter!=V.end();iter++){
       cout<<(*iter)<<endl;

}
```



Principali metodi in STL `vector`

Consultare la documentazione: http://www.cplusplus.com/reference/vector/vector/

Metodo	Descrizione
begin	Restituisce un iteratore che «punta» al primo elemento.
end	Restituisce un iteratore che «punta» ad un elemento *FITTIZIO* finale.
insert	Permette di inserire un elemento in una posizione arbitraria.
erase	Dato un iteratore che «punta» ad un elemento elimina l'elemento corrispondente.
operator[]	Restituisce l'elemento presente in posizione i-esima. *Se la posizione non è valida, causa «undefined behaviour».*
at	Restituisce l'elemento presente in posizione i-esima. *Se la posizione non è valida, segnala un'eccezione.*
size	Restituisce la dimensione.
capacity	Restituisce la capacità.
empty	Restituisce *true* se il vettore è vuoto, *false* altrimenti.
front	Restituisce il primo elemento.
back	Restituisce l'ultimo elemento.
push_back	Inserisce un elemento alla fine.
pop_back	Rimuove l'ultimo elemento.
resize	Dato un intero *n*, ridimensiona il vettore in modo che contenga *n* elementi.
clear	Rimuove tutti gli elementi, ovvero la dimensione diventa 0, la capacità non cambia.

Vettori VS Liste

Liste -> Iteratori Bidirezionali

```
list<string> LS = {"a", "b", "c", "d", "e",};
list<string>::iterator iter = LS.begin();
iter++; // Corretto
iter=iter+1; // Errato
```

Vettori -> Iteratori ad accesso diretto

```
vector<string> VS = {"a", "b", "c", "d", "e",};
vector<string>::iterator iter = VS.begin();
iter++; // Corretto
iter=iter+1; // Corretto
```

Vettori VS Liste

Da preferire i vettori quando:

- È necessario avere accesso diretto agli elementi
- È sufficiente inserire ed eliminare elementi alla fine del vettore

Da preferire le liste quando:

- Non è necessario avere accesso diretto agli elementi
- Non è sufficiente inserire ed eliminare elementi alla fine del vettore
- Non è noto a priori quanti elementi dovranno essere inseriti