



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Standard Template Library (STL) Linked list e il suo iteratore

Stefano Ghidoni



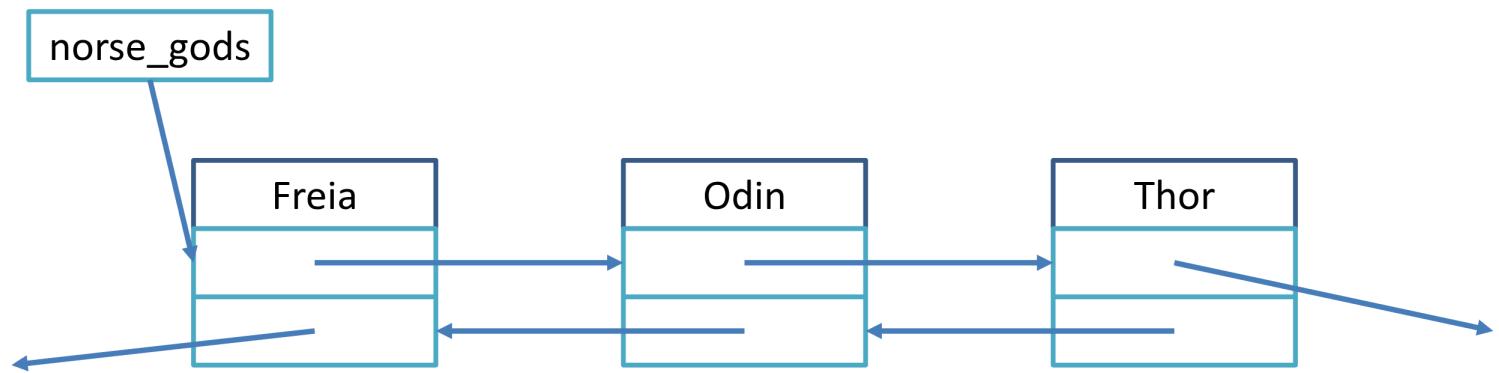
Agenda

- Esempio di implementazione (molto parziale) di linked list STL
- Implementazione degli iteratori
- auto



Linked list

- Una struttura dati gestita così:





Linked list

- Come implementare una linked list?
 - Struct coinvolte:

```
template<typename Elem>
struct Link {
    Link* prev;
    Link* succ;
    Elem val;
};

template<typename Elem> struct list {
    Link<ELEM>* first;
    Link<ELEM>* last;
};
```



Linked list – operazioni

- Operazioni che coinvolgono una list:
 - Quelle di vector (senza []) – costruttore, size, ...
 - Insert ed erase
 - Iteratori
- In STL, un iteratore è un membro della classe container
- Perché è stato escluso []?

L'operator[] è legato all'accesso in tempo costante, che non avviene con una lista (bisogna scorrere tutti gli elementi precedenti/successivi per raggiungere l'elemento desiderato)



Linked list

```
template<typename Elem>
class list {
    // rappresentazione e dettagli implementativi
public:
    class iterator;

    iterator begin();
    iterator end();
```

Restituiscono un iteratore!

```
    iterator insert(iterator p, const Elem& v);
    iterator erase(iterator p);

    void push_back(const Elem& v);
    void push_front(const Elem& v);
    void pop_front();
    void pop_back();

    Elem& front();
    Elem& back();
    // ...
};
```



Iteratore di lista

- Dopo aver progettato list, dobbiamo progettare il suo iteratore
 - Deve contenere un puntatore a un elemento della lista
 - Deve implementare i seguenti operatori:
 - `++`, `--`, `*`, `==`, `!=`



Iteratore di lista

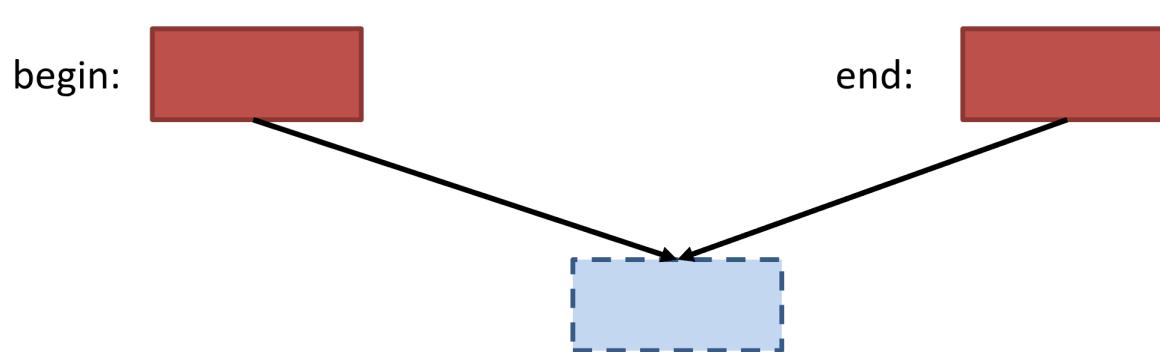
```
template<typename Elemt>
class list<Elemt>::iterator {
    Link<Elemt>* curr;
public:
    iterator(Link<Elemt>* p) : curr{p} {}
    iterator& operator++() { curr = curr->succ;
                                return *this; }
    iterator& operator--() { curr = curr->prev;
                                return *this; }
    Elemt& operator*() { return curr->val; }

    bool operator==(const iterator& b) const {
        return curr == b.curr; }
    bool operator!=(const iterator& b) const {
        return curr != b.curr; }
};
```



Liste vuote

- Se la lista è vuota, $\text{begin} == \text{end}$
 - È la condizione da verificare
- **end che punta a un elemento dopo la fine permette di trattare la lista vuota come un caso *non speciale***





Iteratori e relativo codice

- Gli iteratori tendono a rendere il codice complesso
 - Es:

```
template<typename T>
void user(vector<T>& v, list<T>& lst)
{
    for (vector<T>::iterator p = v.begin(); p != v.end();
         ++p) cout << *p << '\n';

    list<T>::iterator q = find(lst.begin(),
        lst.end(), T{42});
}
```



Iteratori e relativo codice

- Molto spesso il tipo dell'iteratore può essere dedotto dal compilatore:

```
vector<T>::iterator p = v.begin();
```

- Dato che p è inizializzato con v.begin(), non può che essere un vector<T>::iterator



auto

- auto permette di semplificare questa notazione

```
template<typename T>
void user(vector<T>& v, list<T>& lst)
{
    for (auto p = v.begin(); p != v.end();
         ++p) cout << *p << '\n';

    auto q = find(lst.begin();
                  lst.end(); T{42});
}
```



auto

- auto ha un utilizzo più generale:

```
auto x = 123;           // x è un int
auto c = 'y';           // c è un char
auto& r = x;            // r è un int&
auto y = r;             // y è un int (le reference sono
                        // implicitamente dereferenziate)
```

```
// attenzione!
auto s1 = "San Antonio";          // s1 è const char*
string s2 = "Fredericksburg";     // s2 è una string
```

- auto può portare ad ambiguità
 - Da usare con attenzione (consigliato solo per gli iteratori, **sconsigliato/vietato** in tutti gli altri casi!)

Auto ***non*** rende la variabili stile-python. In C++, nella vita di una variabile, il suo tipo non può mutare.



Recap

- Implementazione di list (a partire da link)
- Implementazione dell'iteratore di list
- Uso di auto



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Standard Template Library (STL) Linked list e il suo iteratore

Stefano Ghidoni