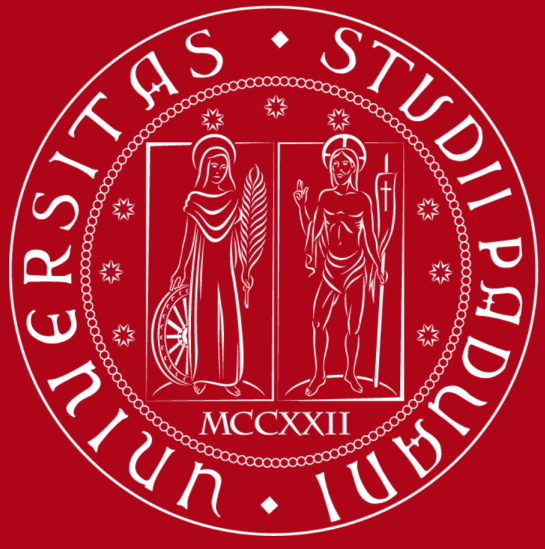


UNIVERSITÀ DEGLI STUDI DI PADOVA

Spostamento di oggetti

Stefano Ghidoni

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- Copia vs move
- Move semantics

Muovere i dati

- Funzione che ritorna un vector

```
vector fill(istream& is)
{
    vector res;
    for (double x; is >> x; ) res.push_back(x);
    return res;
}

void use()
{
    vector vec = fill(cin);
    // uso di vec
}
```

Crea un vettore, interroga infinitamente l'utente, quando viene interrotto il ciclo ritorna il vettore.
La dimensione del vettore può essere molto grande, la decisione non è presa dal programmatore.

- Quale effetto indesiderato è presente?

C'è un problema di efficienza nella restituzione. Si farebbe prima a "rinominare" gli elementi, quindi non vogliamo fare una copia.

Muovere i dati

- "We don't want a copy!" (BS)
- Non possiamo usare res dopo il return
  - Non possiamo usare contemporaneamente res e vec!
- Esiste un modo per spostare i dati, invalidando la sorgente?
  - "We would like to 'steal' the representation of res to use for vec." (BS)
- Ciò che cerchiamo è un'operazione di move
  - Complementa le operazioni di copia

Move semantics

```
class vector {
    int sz;
    double* elem;

public:
    vector(vector&& a);               // move constructor
    vector& operator=(vector&& a);    // move assignment
    // ...
};
```

- La notazione && è chiamata **rvalue reference**
- Gli argomenti non sono const (non possono esserlo!) – perché?
- Le definizioni di operazioni di move tendono a essere più semplici delle copie. Perché?

Viene fatto un intervento forte: si vanno a prendere gli elementi della sorgente e la si invalida

Costruttore e assegnamento move

```
vector::vector(vector&& a)
    : sz{a.sz}, elem{a.elem}
{
    a.sz = 0;           // annulla a
    a.elem = nullptr;
}

vector& vector::operator=(vector&& a)
{
    delete[] elem;
    elem = a.elem;
    sz = a.sz;
    a.elem = nullptr;
    a.sz = 0;
    return *this;
}
```

move viene chiamato quando sta per essere distrutto un elemento e un altro sta per nascere.
Non può avvenire per lo stesso elemento contemporaneamente.
Allora si può chiamare delete come prima operazione.

Fill e use con move constructor

- Riprendendo l'esempio iniziale

```
vector fill(istream& is)
{
    vector res;
    for (double x; is >> x; ) res.push_back(x);
    return res;
}

void use()
{
    vector vec = fill(cin);
    // uso di vec
}
```

Return implementato tramite move constructor

Chiamata al move constructor

- È il compilatore a scegliere tra copy e move constructor
  - Rileva automaticamente le condizioni per usare il move
- Il progettista fornisce lo strumento, il compilatore decide quando usarlo
- Se non è presente il move constructor: copy constructor e distruttore

Move constructor vs copy elision

- In certe circostanze il move constructor non è chiamato
- Sostituito da una tecnica di ottimizzazione del compilatore – **copy elision**
  - L'oggetto è costruito direttamente nella funzione chiamante
- È possibile richiedere al compilatore di non applicare la copy elision
  - Opzione -fno-elide-constructors

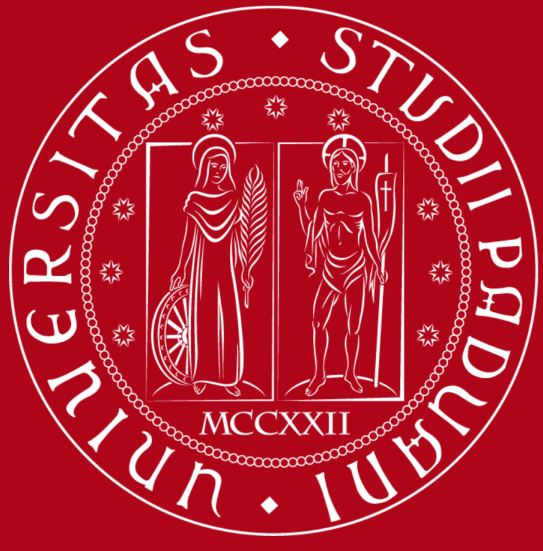
https://medium.com/swlh/c-rvalues-move-semantics-and-copy-elision-36d492da5446
https://en.cppreference.com/w/cpp/language/copy_elision

"Alternativa" a move constructor

- Sappiamo che allocazione dinamica + puntatori permettono ai dati di uscire da una funzione

```
vector* fill2(istream& is)
{
    vector* res = new vector;
    for (double x; is >> x; ) res->push_back(x);
    return res;
}
void use2()
{
    vector* vec = fill2(cin);
    // ... usa vec ...
    delete vec;
}
```

- Questa versione, tuttavia, è molto più verbosa e soggetta a errori



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Spostamento di oggetti

Stefano Ghidoni

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE