

Costruzione di Interfacce Lezione 14 C++STL

cignoni@iei.pi.cnr.it
<http://vcg.iei.pi.cnr.it/~cignoni>

Esempio di template di funzione

Scambio tra due oggetti generici:

```
template <class T> void swap(T &a, T &b){
    T tmp = a;
    a = b;
    b = tmp;
}

int main(){
    int a = 3, b = 16;
    double d = 3.14, e = 2.17;
    swap(a, b);
    swap(d, e);
    // swap (d, a); errore in compilazione!
    return (0);
}
```

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

3

Template

- ❖ I Template sono un meccanismo che permette di definire funzioni e classi basate su argomenti e oggetti dal tipo non specificato

```
template class <T> swap(T &a, T &b);

template class <T> class List (...);
```

- ❖ Queste funzioni e oggetti generici diventano codice completo una volta che le loro definizioni sono usate con oggetti reali

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

2

Esempio di template di classe

Punto 3D generico:

```
template <class T>
class Point3 {
public:
    T v[3];
    Point3 operator + ( Point3 const & p ) const {
        return Point3(v[0]+p.v[0],v[1]+p.v[1],v[2]+p.v[2] );
    }
    Point3 & operator = ( Point3 const & p ){
        v[0]= p.v[0]; v[1]= p.v[1]; v[2]= p.v[2];
        return *this;
    }
};

int main(){
    Point3<float> a1(0,0,0),a2(1,2,3);
    Point3<int> b(1,1,1);
    a1=a1+a2; //ok
    a1=b+a2; //error!!
    return (0);
}
```

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

4

Template Osservazioni

- ❖ Notare che la definizione di una funzione template è simile ad una macro, nel senso che la funzione template *non è ancora codice*, ma lo diventerà una volta che essa viene usata
- ❖ Il fatto che il compilatore generi codice concreto solo una volta che una funzione è usata ha come conseguenza che una template function non può essere mai raccolta in una libreria a run time
- ❖ Un template dovrebbe essere considerato come una sorta di dichiarazione e fornito in un file da includere.
- ❖ Quando da un template si genera codice per un certo specifico tipo si dice che si è istanziato quel template per quel tipo

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

5

Namespace

- ❖ Lo spazio dei nomi delle classi e delle funzioni globali è unico.
- ❖ Per questo motivo i venditori di librerie e di classi di solito danno nomi lunghi e con vari prefissi che li rendano non ambigui (e.g. SQL_Acc_Start(...))
- ❖ In c++ lo spazio globale dei nomi è scomponibile in blocchi usando i *namespace*

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

6

Namespace

```
namespace MyLib
{
class MyClass {...};
}
int main()
{
myLib::MyClass p;
}
```

❖ Nota

- ❖ Namespace possono apparire solo nello scope globale (non dentro una classe o una funz),
- ❖ possono essere nested
- ❖ Un namespace puo *continuare* in file diversi (non è quindi una ridefinizione).
- ❖ Non ci vuole il ';' in fondo

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

7

Namespace use

- ❖ Per accedere ai nomi definiti in un namespace

❖ Direttiva using

```
using namespace myLib;
myClass p
```

❖ Risolutore di Scope ::

```
myLib::myClass p
```

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

8

STL: Standard Template Library

- ❖ Libreria di classi container, algoritmi, iteratori che mette a disposizione in forma astratta e generica (basata su template) la maggior parte degli algoritmi e strutture dati "classici"
 - ❖ Liste, insiemi ordinati, vettori, hash, code di priorità, stack ecc.
 - ❖ Sort, permutazioni, ricerche binarie, ecc
- ❖ Tutte le entità della STL sono definite nel namespace 'std'

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

9

STL Container

- ❖ I container sono tipi di dato astratti in cui è possibile memorizzare e ritrovare informazione.
- ❖ Templated sul tipo dell'oggetto da memorizzare
- ❖ Possono essere sequenziali, ad accesso casuale, associativi, ordinati ecc...

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

10

STL Containers

- ❖ Tutti i container supportano:
 - ❖ overloaded assignment operator
 - ❖ Tests uguaglianza: == and !=
 - ❖ Ordering operators: <, <=, > and >=.L'operatore < ritorna true se ogni elemento del primo e minore del corrispondente elemento del secondo container.
- ❖ Per poter istanziare un contenitore su di un certo tipo esso deve avere:
 - ❖ A default-value (e.g., a default constructor)
 - ❖ The equality operator (==)
 - ❖ The less-than operator (<)

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

11

pair

- ❖ Usato per memorizzare una coppia di elementi senza aver bisogno di fare una classe apposita.

```
#include <utility>
using namespace std;
pair<string, string>
piper("PA28", "PH-ANI"),
cessna("C172", "PH-ANG");
cout << piper.first << endl << // shows 'PA28'
      cessna.second << endl; // shows 'PH-ANG'
```

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

12

Vector

- ❖ Implementa un array resizable
- ❖ Costruttori tipici e init
- ❖ `#include <vector>`
- ❖ `vector<int> A;`
- ❖ `A.resize(10);`
- ❖ `vector<int> B(4);`
- ❖ `vector<float> C(10,0.1f);`

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

13

vector

- ❖ Accesso
- ❖ `Operator[]`
- ❖ `A[3]=5; // Nota: è responsabilità dell'utente non uscire dal vettore;`
- ❖ `A.back()` è un riferimento all'ultimo elemento
- ❖ `A.begin()` iteratore al primo
- ❖ `A.end()` iteratore al primo dopo l'ultimo

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

14

iteratori

- ❖ Astrazione del concetto di puntatore.
- ❖ Sempre riferiti ad un istanza di un certo container:
 - ❖ `vector<int>::iterator it;`
 - ❖ `list<myClass *>::iterator li;`
- ❖ Dato un iteratore iter
 - ❖ *iter rappresenta l'oggetto a cui l'iteratore punta
 - ❖ ++iter fa avanzare l'iteratore nel contenitore in maniera ragionevole
 - ❖ Se il contenitore è di tipo con accesso casuale vale anche l'aritmetica dei puntatori (iter+10 avanza di 10 posizioni)
 - ❖ Usati di solito per indicare range in un container in maniera inclusiva a sx
[first,last) indica un range di elementi in un container da first (incluso) a last (escluso);

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

15

Tipico uso iteratori

- ❖ Scansione di un container con iteratori

```
vector<myClass> V;
...
vector<myClass>::iterator vi;
for (vi=V.begin(); vi!=V.end(); ++vi)
    doSomething(*vi);

list<myClass> L;
...
list<myClass>::iterator li;
for (li=L.begin(); li!=L.end(); ++li)
    doSomething(*li);
```

- ❖ Sempre nello stesso modo per tutti i container!

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

16

Tipico uso iteratori

- ❖ Definizione di un range con iteratori

```
vector<myClass> V;
sort(V.begin(), V.end());

reverse(V.begin(), V.end());
```

- ❖ Nota [first,last) indica un range di elementi in un container da first (incluso) a last (escluso);
- ❖ end() è il primo oltre la fine del container e quindi è sempre non accessibile!!!

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

17

vector

- ❖ Aggiunta di elementi nel vettore

```
Vector<myObj> V;
myObj obj;
V.push_back(obj);
```

- ❖ Nota:

- ❖ Costo: ammortizzato costante;
- ❖ L'allocazione di memoria è gestita in maniera trasparente (fa il resize quando serve raddoppiando la size).
- ❖ Il vettore *contiene* gli oggetti, quindi si memorizza in V una copia di obj

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

18

vector

- ❖ Elementi possono essere aggiunti anche in altre posizioni ma con costo lineare nella grandezza del vettore;
- ❖ Operatori utili
 - ❖ `resize(int)` (nota che non disalloca...)
 - ❖ `reserve(int)`
 - ❖ `capacity()`

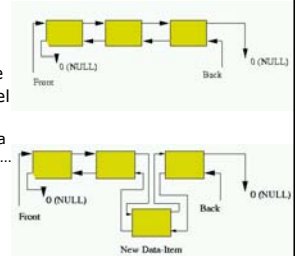
8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

19

list

- ❖ Il container list implementa una generica double linked list
- ❖ Operazioni di inserzione, cancellazione in posizione qualsiasi in tempo costante
- ❖ Persistenza degli oggetti nel container in memoria
 - ❖ Iteratori agli elementi della lista restano validi a lungo...



8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

20

map

- ❖ Implementa un array associativo ordinato
- ❖ `map<key, value> object;`
- ❖ Permette di accedere ad un insieme di oggetti per contenuto.
- ❖ `operator[]` ridefinito per accedere per key

```
map<string,int> si;
si["pippo"]=42;
```

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

21

Altri container utili

- ❖ queue
- ❖ set
- ❖ stack
- ❖ hash e hashmap

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

22

Esempio di classe template

- ❖ Classe per la rappresentazione generica di un punto in uno spazio 3D, templatato sul tipo usato per tenere I valori delle coordinate.
- ❖ `Point3<float>`
- ❖ Overload di tutti gli operatori sensati.

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

23

STL algorithms

- ❖ Sui container visti finora nelle stl sono definiti un lunga serie di algoritmi generici che permettono di fare azioni comuni piu' o meno semplici.

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

24

STL algorithms: Non mutating

1. [for_each](#)
2. [find](#)
3. [find_if](#)
4. [adjacent_find](#)
5. [find_first_of](#)
6. [count](#)
7. [count_if](#)
8. [mismatch](#)
9. [equal](#)
10. [search](#)
11. [search_n](#)
12. [find_end](#)

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

25

STL algorithms: Mutating

- | | |
|------------------------------------|--------------------------------------|
| 1. copy | 1. Remove |
| 2. copy_n | 1. remove |
| 3. Swap | 2. remove_if |
| 1. swap | 3. remove_copy |
| 2. iter_swap | 4. remove_copy_if |
| 3. swap_ranges | 2. unique |
| 4. transform | 3. unique_copy |
| 4. Replace | 4. reverse |
| 1. replace | 5. reverse_copy |
| 2. replace_if | 6. rotate |
| 3. replace_copy | 7. rotate_copy |
| 4. replace_copy_if | 8. random_shuffle |
| 5. fill | 9. random_sample |
| 6. fill_n | 10. random_sample_n |
| 7. generate | 11. partition |
| 8. generate_n | 12. stable_partition |

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

26

STL algorithms: Sorting

- | | |
|--------------------------------------|---|
| 1. Sort | 1. Set operations on sorted ranges |
| 1. sort | 1. includes |
| 2. stable_sort | 2. set_union |
| 3. partial_sort | 3. set_intersection |
| 4. partial_sort_copy | 4. set_difference |
| 5. is_sorted | 5. set_symmetric_difference |
| 2. nth_element | 2. Heap operations |
| 3. Binary search | 1. push_heap |
| 1. lower_bound | 2. pop_heap |
| 2. upper_bound | 3. make_heap |
| 3. equal_range | 4. sort_heap |
| 4. binary_search | 5. is_heap |
| 4. merge | 3. Minimum and maximum |
| 5. inplace_merge | 1. min |
| | 2. max |
| | 3. min_element |
| | 4. max_element |

8 Nov 2002

Costruzione di Interfacce - Paolo Cignoni

27