Costruzione di Interfacce
Lezione 12
C++STL

cignoni@isti.cnr.it
http://vcg.isti.cnr.it/~cignoni

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

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);
}
```


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);
}
```


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

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*

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

Namespace use

- Per accedere ai nomi definiti in un namespace
  - Direttiva using

```
Using namespace myLib;
myClass p
```

  - Risolutore di Scope ::

```
myLib::myClass p
```


STL: Standard Template Library

- Libreria di classi container, algoritmi, iteratori che mette a disposizone 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'

STL Container

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

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 corrispndente 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 (<)

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'
```

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);
```


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

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);

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(*ii);

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!

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 accedibile!!!

vector

- Aggiunta di elementi nel vettore

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

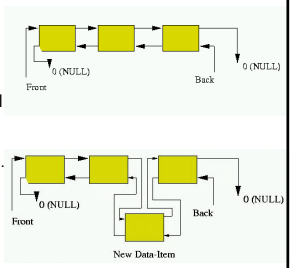
- Nota:
  - Costo: ammortizzato costante;
  - L'allocazione di memoria è gestita in maniera trasparente (fa il resize quando serve raddoppiando la size).
  - Il vettore *contiene* gli oggetti, quindi si memorizza in V una copia di obj

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()`

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…



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;
```


Altri container utili

- queue
- set
- stack
- hash e hashmap

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.

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.

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

STL algorithms: Mutating

1. copy
2. copy_n
3. Swap
   1. swap
   2. iter_swap
   3. swap_ranges
   4. transform
4. Replace
   1. replace
   2. replace_if
   3. replace_copy
   4. replace_copy_if
5. fill
6. fill_n
7. generate
8. generate_n

1. Remove
   1. remove
   2. remove_if
   3. remove_copy
   4. remove_copy_if
2. unique
3. unique_copy
4. reverse
5. reverse_copy
6. rotate
7. rotate_copy
8. random_shuffle
9. random_sample
10. random_sample_n
11. partition
12. stable_partition

STL algorithms: Sorting

1. Sort
   1. sort
   2. stable_sort
   3. partial_sort
   4. partial_sort_copy
   5. is_sorted
2. nth_element
3. Binary search
   1. lower_bound
   2. upper_bound
   3. equal_range
   4. binary_search
4. merge
5. inplace_merge

1. Set operations on sorted ranges
   1. includes
   2. set_union
   3. set_intersection
   4. set_difference
   5. set_symmetric_difference
2. Heap operations
   1. push_heap
   2. pop_heap
   3. make_heap
   4. sort_heap
   5. is_heap
3. Minimum and maximum
   1. min
   2. max
   3. min_element
   4. max_element