Sperimentazioni di Fisica I mod. A – Laboratorio 5

Vettori ed Iteratori

Dipartimento di Fisica e Astronomia "G. Galilei",
Università degli Studi di Padova

Contenitori sequenziali: `std::vector`

- la classe `std::vector` permette di creare un array dinamico e di accedere o manipolare un suo elemento data la posizione (indice) usando l'operatore `[]`

```
vector<double> vd; //crea un vettore di double
```

1. è un array dinamico
2. è possibile aggiungere o togliere elementi, modificando run-time la dimensione
3. normalmente si aggiungono elementi nuovi alla fine, ma è anche possibile inserirli all'inizio o in una posizione qualsiasi
4. è possibile effettuare delle ricerche all'interno del vector

Es 1 - creazione di vettori

```
#include <vector>
int main()
{
  // Un vettore dinamico
       vector <double> v_dyna;

  // Un vettore con 10 elementi
       vector <int> vint_10_elements(10);

  // Un vettore con 10 elementi, inizializzati a 90
       vector <int> vint_10(10, 90);

  // Un vettore inizializzato con il contenuto di un altro
       vector<int> vc1(vint_10_elements);
       vector<int> vc2 = vint_10_elements;

  // Un vettore con 5 valori presi da un altro
       vector<int> vcopy_5(vint_10.begin(),vint_10.begin()+5);
}
```

Gli iteratori nella STL

- l'esempio più semplice di iteratore è un puntatore
- Gli iteratori sono classi template che possono essere pensati come una generalizzazione dei puntatori
- tramite gli iteratori è possibile *navigare* tra gli elementi di un contenitore in modo da accedere agli elementi ed eseguire operazioni
- gli iteratori STL posssono essere classificati in
  - Iteratore di Input : possono essere usati per accedere ad un oggetto che si trova in una collezione.
  - Iteratori di Output : permettono al programatore di scrivere nella collezione.
- inoltre, a seconda di come permettono di muoversi nella collezione sono
  - Iteratori in Avanti
  - Iteratori Bidirezionali
  - Iteratori ad Accesso Casuale

Es 2 - Iteriamo su un vettore

```
#include <iostream>
#include <vector>

int main()
{
  // Fissiamo la dimensione dell'array e inizializziamolo
       vector<double> v(4) = {1.5, 2.3, 3.3, 4.8};

  // Iteriamo, e calcoliamo il massimo
  v_max = v.at(0);
  for (auto e : v)
      if (e > v_max)  v_max = e;

       cout << "Il vettore contiene ";
       cout << v.size() << " elementi\n";
       cout << "Valore massimo: " << v_max <<       endl;

}
```

Es 3 - Uso degli iteratori

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
  vector <double> v(4);
  ...

  vector <double>::iterator walk = v.begin();
  while (walk != v.end())
  {
    cout << * walk << endl;
    // L'iteratore va incrementato
    // per accedere all'elemento successivo
    walk++;
  }

}
```

Es 4 - Uso della sintassi degli array

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
  vector <double> v(4);
  ...
  int index = 0;
  while (index < v.size())
  {
    cout << v.at(index) << endl;
    // Il contatore va incrementato
    // per accedere all'elemento successivo
    index++;
  }

}
```

Es 5 - Ricerca del numero 2991 nel vettore

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
  vector <int> vi;
  ...
  // element e di tipo vector <int>::iterator
  auto element = find(vi.begin(), vi.end(), 2991)

  // Controlliamo se l'elemento e' stato trovato
  if (element != vi.end())
  {
      int index = distance(vi.begin(), element);
      cout << "Valore " << *element;
      cout << " trovato alla posizione " << index << endl;
  }

}
```

Es 6 - Rimozione di un elemento alla fine del vettore

```
#include <iostream>
#include <vector>
int main()
{
  // Un vettore dinamico
       vector <int> v_dyna;

  // Inseriamo dei valori
  v_dyna.push_back(23);
  v_dyna.push_back(76);

       cout << "Il vettore contiene ";
       cout << v_dyna.size() << " elementi\n";

  // Estraiamo un valore
  v_dyna.pop_back();

       cout << "Il vettore contiene ";
       cout << v_dyna.size() << " elementi\n";
}
```

std::vector - costruttori

```
vector <type> v
```

- crea un vettore di tipo `type`

```
vector <type> v(n)
```

- crea un vettore di tipo `type` e dimensione `n`

```
vector <type> v(n, const T & t)
```

- crea un vettore di tipo `type` con dimensione `n` e lo inizializza ai valori costanti `t`

```
vector <type> v(begin_iterator,end_iterator)
```

- crea un vettore di tipo `type` e lo inizializza copiando gli elementi da un altro vettore da `begin_iterator` a `end_iterator`

std::vector - metodi

`v.begin()`

- (iteratore) : ritorna un iteratore che punta al primo elemento del vettore

`v.end()`

- (iteratore) : ritorna un iteratore che punta all'elemento successivo all'ultimo

`v.size()`

- (int) : ritorna il numero di elementi del vettore (è uguale a `v.end() - v.begin()`)

`v.capacity()`

- (int) : ritorna il numero di elementi inseribili senza necessità di riallocare la memoria

`a == b`

- (bool) : ritorna `true` se `a` e `b` hanno la stessa dimensione e ogni elemento di `a` è equivalente (`==`) al corrispondente elemento di `b`

std::vector - metodi

`v.push_back(t)`

- (void) : inserisce un elemento `t` alla fine del vettore

`v.pop_back()`

- (void) : elimina l'ultimo elemento inserito nel vettore

`v.front()`

- (T &) : ritorna il primo elemento

`v.back()`

- (T &) : ritorna l'ultimo elemento

`v.[i]`

- (T &) : accede all'elemento `i`

`v.at(i)`

- (T &) : accede all'elemento `i` (e contolla che i non superi i limiti degli indici dell'array - bound checking)

`v.clear()`

- (void) : rimuove tutti gli elementi dal vettore

Addendum

```
int n;
vector <double> v(n); \\ Dimensione del vettore ignota.
cin >> n;

vector <double> v; \\ Dimensione del vettore 0!
double a;
while(cin >> a)
      v.push_back(a);

int n; cin >> n;
vector <double> v(n); \\Dimensione del vettore n
for(int i = 0; i < n; ++i)
      cin >> v[i]; \\ oppure (cin >> v.at(i));
\\ Metodo Alterativo
for( auto & x : v )
      { cin >> a ;    \\ NO v.at(x) NO v[x]
        x = a;        \\ Se il vector ha dimensione n
      }               \\ push_back scrive elemento n+1
```

Comandi per Compilare

Comando generale:

```
g++
```

Per utilizzare le features dello standard C++11 :

- dalla versione GCC 4.7 in poi:

```
g++ -std=c++11
```

- per le versioni precedenti:

```
g++ -std=c++0x
```