Un assaggio di STL con i vettori

Nell'insegnamento non si può vedere il frutto di una giornata di lavoro. È invisibile e rimane così, forse per venti anni.

Jacques Barzun

Si dice che i programmatori siano pigri. Di sicuro, se qualcuno ha già lavorato per noi, non si vede il motivo per rifare lo stesso lavoro una seconda volta. Uno di questi "qualcuno", nel contesto del C++, si chiama **Alexander Stepanov**, il quale è il principale artefice di una specifica libreria di strumenti realizzata appositamente nella visione della programmazione generica: la **Standard Template Library**, meglio conosciuta con l'acronimo di **STL**.

Alexander Alexandrovich Stepanov, il principale artefice della Libreria STL.

Volendo schematizzare al massimo la libreria STL possiamo vederla come composta di tre elementi:

- Contenitori
- Iteratori
- Algoritmi

I **contenitori,** detti anche in modalità anglosassone **containers**, sono i veri e propri aggregati di dati. Si tratta quindi di collezioni organizzate di elementi. A scopo di esempio pensate alla struttura coda di cui vi ho scritto pocanzi.
Gli **iteratori** sono degli speciali puntatori che consentono di navigare, cioè di muoversi, all'interno delle strutture dati.
Gli **algoritmi** sono delle procedure preconfezionate per risolvere i classici problemi che si presentano in relazione alle strutture dati: ordinamento di dati, ricerca di specifici valori, ecc.
Per cercare di essere minimamente concreti di seguito vi propongo un codice minimale relativo all'uso dei vettori. Sebbene il termine vettore venga spesso considerato come sinonimo di array, un vettore (vector) è una struttura dati di tipo parametrico che può essere lavorata in maniera dinamica in quanto, contrariamente ad un array che viene gestito nell'area di memoria stack, esso vive nella sezione di memoria heap.

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

using namespace std; //oppure devi usare istruzioni del tipo std::vector<int> v;

int main(int argc, char** argv)
{
    printf("I vettori (vectors) non sono array ;)\n");
    //Dichiaro un vector (le parentesi angolari sono giustificate dal fatto che si tratta di una classe
template)
    //Un vector consente la gestione di memoria contigua peremettendo di riferirsi ad essa in un modo del
tipo: v[0], v[1]...
    int n = 10;
    vector<int> array(n); //creaiamo un vector per 10 interi (vengonoinizialittati a 0)
    //verifichiamo il suo contenuto
    printf("Contenuto vector: ");
    for(int i=0; i<n; ++i)
    {
        printf("%d ", array[i]);
    }
    printf("\n");

    // valorizziamo il vector
    for(int i=0; i<n; ++i)
    {
        array[i] = i;
    }

    //verifichiamo il suo contenuto
    printf("Contenuto vector: ");
    for(int i=0; i<n; ++i)
    {
        printf("%d ", array[i]);
    }
    printf("\n");

    //In ogni caso NON abbiamo necessità di delete

    //inserisco alla fine dell'array un nuovo elemento intero con valore 66
    array.push_back(66);

    //verifichiamo il suo contenuto
    //Usiamo ora la il metodo size() per determinare la nuova dimensione del vector
    printf("Contenuto vector: ");
    for(int i=0; i<array.size(); ++i)
    {
        printf("%d ", array[i]);
    }
    printf("\n");
```

```
    //ora voglio ridimensionare il vector eliminando così il valore inserito
    int temp=array.size(); //determino l'attuale dimensione del vector
    array.resize(temp-1); //quindi ridimensione il vector alla dimensione attuale meno 1

    //verifichiamo il contenuto del vettore
    printf("Contenuto vector: ");
    for(int i=0; i<array.size(); ++i)
    {
        printf("%d ", array[i]);
    }
    printf("\n");

    //Un altro modo per eliminare l'ultimo elemento del vettore
    array.pop_back();
    printf("Contenuto vector: ");
    for(int i=0; i<array.size(); ++i)
    {
        printf("%d ", array[i]);
    }
    printf("\n");


    return 0;
}
```