Class Vector C++

In C++, una "**vector**" si riferisce generalmente alla classe std::vector, che è un contenitore di array dinamico fornito dalla libreria standard di C++.
Un **vector** consente di creare un array ridimensionabile, simile a un array in molti altri linguaggi di programmazione, ma con funzionalità aggiuntive come il ridimensionamento automatico, la gestione dinamica della memoria e una vasta gamma di funzioni

Inclusione dell'Intestazione: Per utilizzare std::vector, è necessario includere l'intestazione <vector>:

```
#include <vector>
```

Dichiarazione e Inizializzazione: È possibile dichiarare un std::vector e inizializzarlo con elementi specifici o una dimensione iniziale:

```
std::vector<int> mioVector;            // Dichiarazione di un vettore vuoto di interi.
std::vector<double> doubleVector(5);   // Dichiarazione di un vettore di 5 double, inizializzato a 0.0.
std::vector<std::string> strVector = {"Ciao", "Mondo"}; // Dichiarazione e inizializzazione di un vettore di stringhe.
```

Aggiunta di Elementi: È possibile aggiungere elementi al vettore utilizzando `push_back()`

```
std::vector<int> mioVector;
mioVector.push_back(42);    // Aggiunge l'intero 42 al vettore.
```

Accesso agli Elementi: E' possibile accedere agli elementi del vettore utilizzando l'operatore di accesso agli indici [] o la funzione at().
Ad esempio:

```
std::vector<int> myVector = {10, 20, 30, 40};
int firstElement = myVector[0];  // Accede al primo elemento (10).
int secondElement = myVector.at(1); // Accede al secondo elemento (20).
```

Dimensione: La funzione size() restituisce il numero di elementi effettivi nel vettore.

```
std::vector<int> myVector = {1, 2, 3};
int size = myVector.size();      // Restituisce 3.
```

Modifica degli Elementi: Puoi modificare gli elementi del vettore direttamente assegnando nuovi valori:

```
std::vector<int> myVector = {1, 2, 3};
myVector[1] = 42;  // Modifica il secondo elemento (2 diventa 42).
```

Rimozione di Elementi: E' possibile rimuovere elementi dal vettore utilizzando il metodo erase() o pop_back():

```
std::vector<int> myVector = {1, 5, 7, 8};
myVector.erase(2);  // Rimuove il secondo elemento.
myVector.pop_back();  // Rimuove l'ultimo elemento.
```

Iterazione: E’ possibile iterare attraverso i contenuti di un vettore utilizzando un ciclo for:

```
std::vector<int> myVector = {1, 2, 3};
for (int i = 0; i < myVector.size(); ++i) {
                // Fai qualcosa con myVector[i].
```

Inserimento di Elementi: E’ possibile inserire elementi in un vettore in varie posizioni utilizzando il metodo insert():

```
std::vector<int> myVector = {1, 2, 3};
myVector.insert(1, 42); // Inserisce 42 alla posizione 1.
```

Svuotare un Vettore: E’ possibile svuotare completamente un vettore utilizzando il metodo clear():

```
std::vector<int> myVector = {1, 2, 3};
myVector.clear();  // Svuota il vettore (size diventa 0).
```

Allocazione Dinamica della Memoria: Un grande vantaggio di std::vector è che gestisce automaticamente l'allocazione e la deallocazione dinamica della memoria. Non è necessario preoccuparsi della gestione manuale della memoria.

Performance: std::vector è generalmente efficiente per l'accesso casuale agli elementi e l'aggiunta/rimozione di elementi dalla fine del vettore. Tuttavia, l'aggiunta/rimozione di elementi dal principio o dal centro del vettore può essere costosa in termini di prestazioni, poiché richiede il ridimensionamento e la copia degli elementi.

Nota: se si usa il

```
using namespace std;
```

L'istruzione

```
std::vector<int> myVector = {1, 2, 3};
```

si può riscrivere come

```
vector<int> myVector = {1, 2, 3};
```