Classe: vettore di interi

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

int main() {

    VettoreInteri vett;

    ...

    return 0;
}
```

Oggetto VettoreInteri contenente
10 elementi di tipo intero

- Se volessimo un vettore di 20 elementi?
- Se volessimo un vettore di 25 elementi?
- Se volessimo decidere la dimensione del vettore sulla base di un numero letto da tastiera?

Classe: vettore di interi

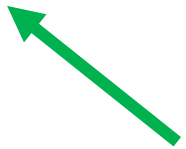
```
#include <iostream>
using namespace std;

const int MAX = 100;
class VettoreInteri {

   int v[MAX];
   int dim;

public:

   VettoreInteri(int n=MAX);

   bool inserisci(int indice, int valore);
   bool ottieni(int indice, int& valore);
   int cerca(int valore);
   int conta(int valore);
   bool simmetrico();
   void inverti();
   void trasla();
};
```

Possibile soluzione (non ottimale):
Sovradimensionamento del vettore

Campo dati che indica la dimensione effettiva del vettore
Inizializzato dal costruttore

Classe: vettore di interi

```
VettoreInteri::VettoreInteri(int n) {
    if (n > 0 && n < MAX)
        dim = n;
    else
        dim = MAX;
    for (int i=0; i<dim; i++)
        v[i] = 0;
}

bool VettoreInteri::inserisci(int indice, int valore) {

    if (indice < 0 || indice >= dim)        // controllo validita' parametri
        return false;

    v[indice] = valore;
    return true;
}

...
```



Tutte le funzioni membro considerano **dim** come dimensione massima del vettore, non **MAX**

Matrici all'interno della classe

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

const int R = 3;
const int C = 4;
class MatriceInteri {

   int m[R][C];

public:
   ...
};

int main() {

    MatriceInteri mat;

    ...

    return 0;
}
```

Matrice con 3 righe e 4 colonne
Ogni elemento è di tipo intero

Dichiarazione oggetto di tipo MatriceInteri

Classe: matrice di interi

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

const int R = 3;
const int C = 4;
class MatriceInteri {

   int m[R][C];

public:

   MatriceInteri();
   bool inserisci(int r, int c, int valore);
   bool ottieni(int r, int c, int& valore);
   bool cerca(int& r, int& c, int valore);
   int conta(int valore);
   void serpentina();
};
```

Classe: matrice di interi

```
MatriceInteri::MatriceInteri() {
    for (int i=0; i<R; i++)
        for (int j=0; j<C; j++)
            m[i][j] = 0;
}
bool MatriceInteri::inserisci(int r, int c, int valore) {
    if (r < 0 || r >= R)        // controllo validita' parametri
        return false;
    if (c < 0 || c >= C)
        return false;
    m[r][c] = valore;
    return true;
}
bool MatriceInteri::ottieni(int r, int c, int& valore) {
    if (r < 0 || r >= R)        // controllo validita' parametri
        return false;
    if (c < 0 || c >= C)
        return false;
    valore = m[r][c];
    return true;
}
```

Classe: matrice di interi

```
bool MatriceInteri::cerca(int& r, int& c, int valore) {
    for (int i=0; i<R; i++) {
        for (int j=0; j<C; j++) {
            if (m[i][j] == valore) {
                r=i;  c=j;
                return true;
        }
    }
    return false;
}

int MatriceInteri::conta(int valore) {
    int cont = 0;
    for (int i=0; i<R; i++) {
        for (int j=0; j<C; j++) {
            if (m[i][j] == valore)
               cont++;
        }
    }
    return cont;
}
```

Classe: matrice di interi

```
void MatriceInteri::serpentina() {
    int cont = 0;
    for (int i=0; i<R; i++) {
        if (i%2 == 0) {
            for (int j=0; j<C; j++) {
                m[i][j] = cont;
                cont++;
            }
        }
        else {
            for (int j=C-1; j>=0; j--) {
                m[i][j] = cont;
                cont++;
            }
        }
    }
}
```

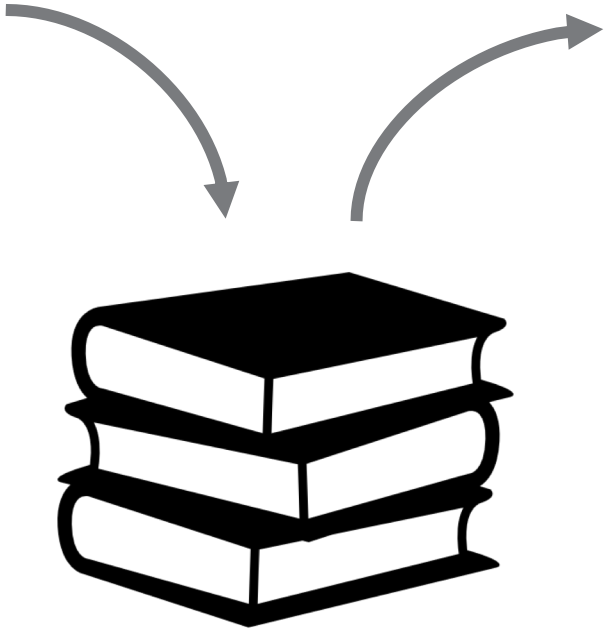
Tipo di dato Pila

Insieme di elementi, dove l'inserimento e l'estrazione di un elemento si effettua da un solo punto

Politica **LIFO**: **L**ast **I**n **F**irst **O**ut

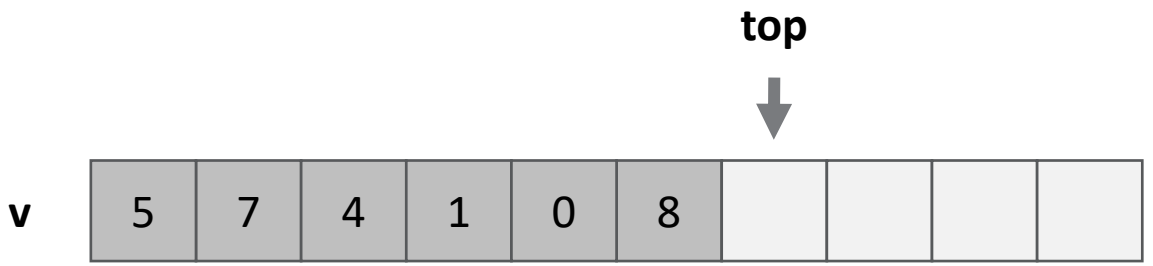
Esempi:

- Pila di libri
- Persone che entrano in un ascensore

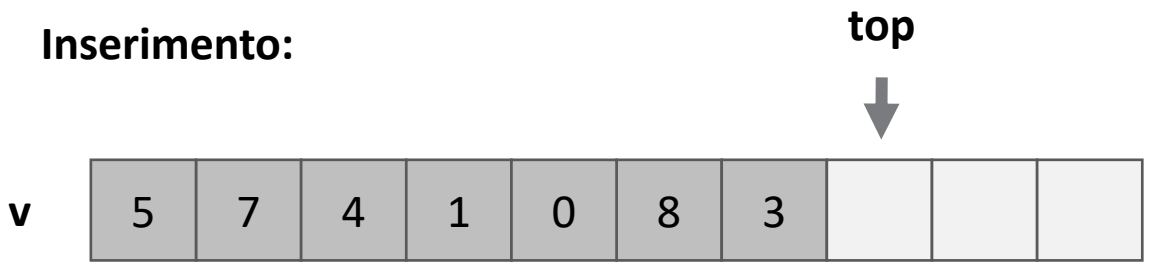


Classe: Pila

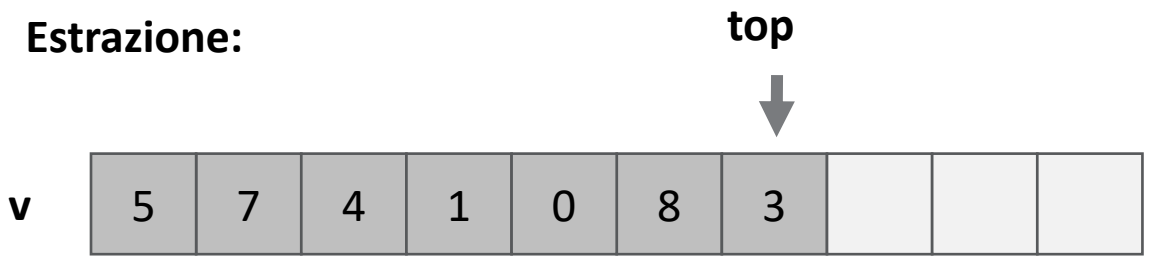
```
#include <iostream>
using namespace std;

const int DIM = 10;

class Pila {

    int v[DIM];
    int top;

public:

    Pila();
    bool inserisci(int valore);
    bool estrai(int& valore);
    bool piena();
    bool vuota();
};
```



Inserimento:



Estrazione:



Classe: Pila

```
Pila::Pila() {
    top = 0;
}

bool Pila::vuota() {
    if (top == 0)
        return true;
    return false;
}

bool Pila::piena() {
    if (top >= DIM)
        return true;
    return false;
}
```

```
bool Pila::inserisci(int valore) {
    if (piena())
        return false;
    v[top] = valore;
    top++;
    return true;
}

bool Pila::estrai(int& valore) {
    if (vuota())
        return false;
    top--;
    valore = v[top];
    return true;
}
```

Tipo di dato Coda

Insieme di elementi, dove l'inserimento e l'estrazione di un elemento si effettuano dai punti opposti

Politica **FIFO**: **F**irst **I**n **F**irst **O**ut

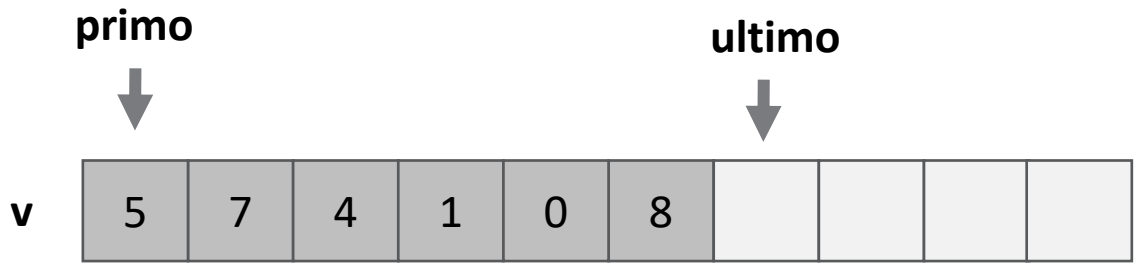
Esempi:

- Sportello bancario
- Cassa del supermercato
- Semaforo

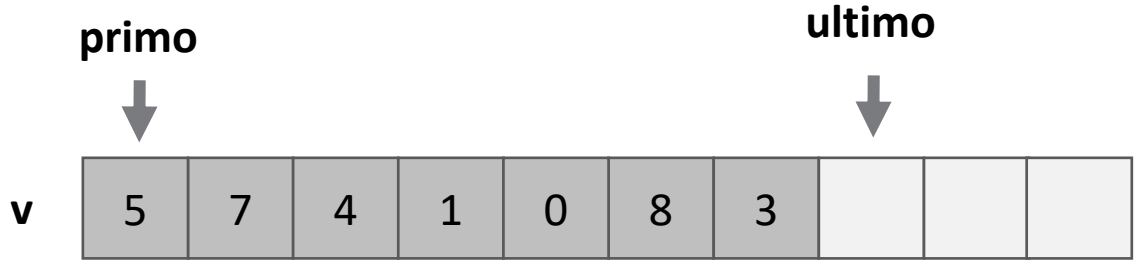
Classe: Coda

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

const int DIM = 10;

class Coda {

    int v[DIM];
    int primo, ultimo;

public:

    Coda();
    bool inserisci(int valore);
    bool estrai(int& valore);
    bool vuota();
    bool piena();
};
```

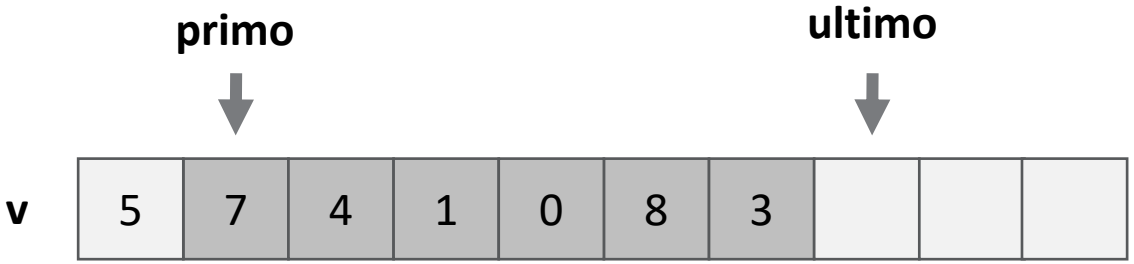
Sfrutta un array gestito in modo *circolare*



Inserimento:

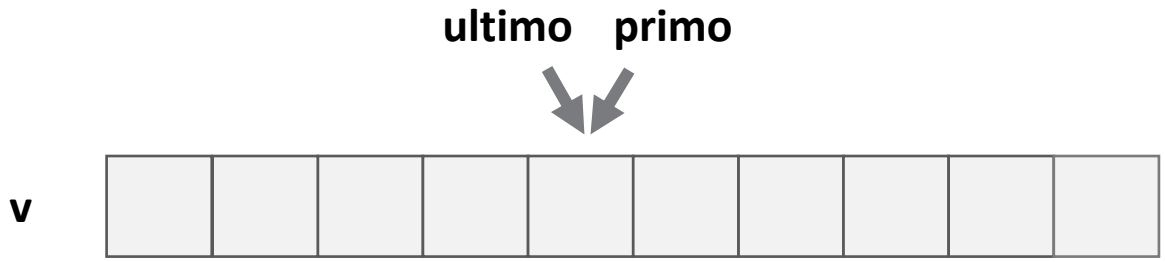


Estrazione:

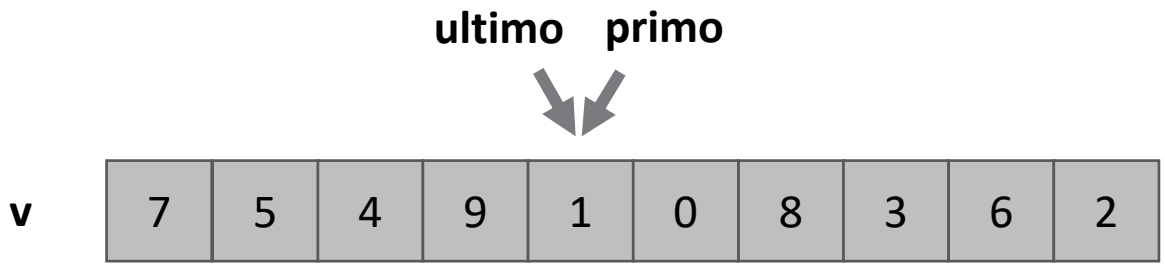


Classe: Coda

Quanti elementi posso realmente inserire nella coda?



Coda vuota → primo == ultimo

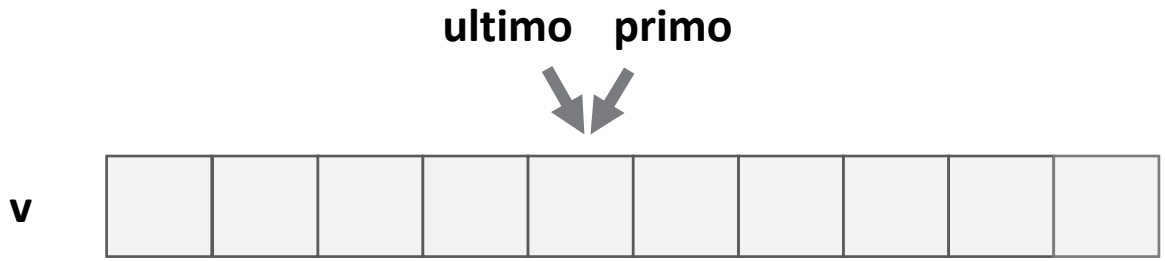


Coda piena → primo == ultimo

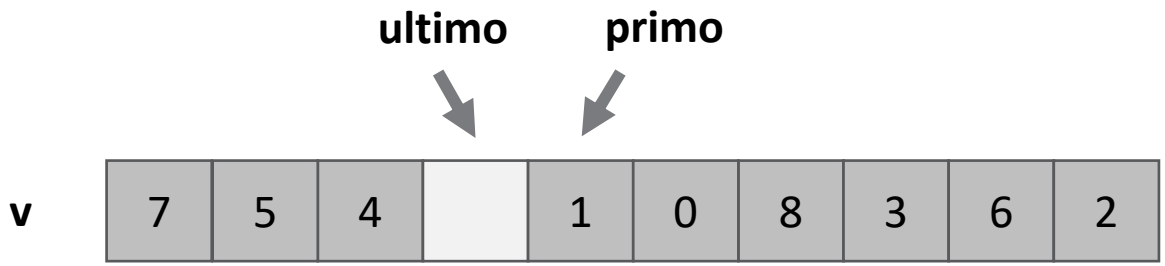
Se primo == ultimo, come posso decidere se la coda è piena o vuota?

Classe: Coda

Quanti elementi posso realmente inserire nella coda?



Coda vuota → primo == ultimo



Coda piena → primo == (ultimo+1)%10

Con questa rappresentazione, posso inserire solo 9 elementi nella coda!

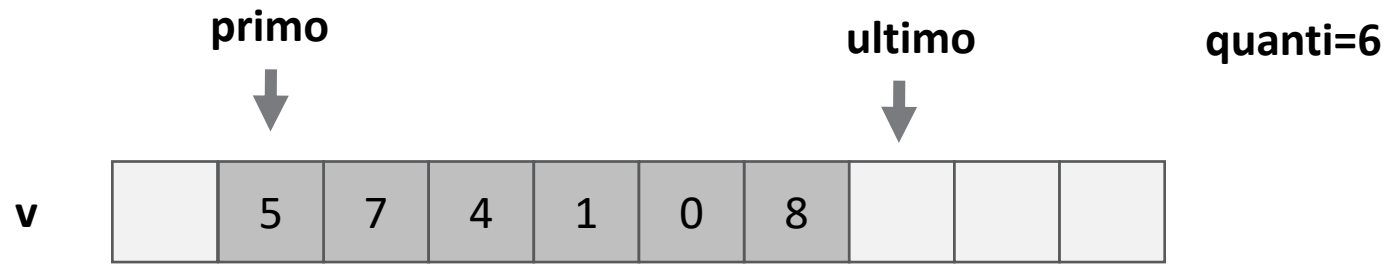
Classe: Coda

```
Coda::Coda() {
    primo = 0;
    ultimo = 0;
}

bool Coda::vuota() {
    if (primo == ultimo)
        return true;
    return false;
}

bool Coda::piena() {
    if (primo == (ultimo+1)%DIM)
        return true;
    return false;
}
```

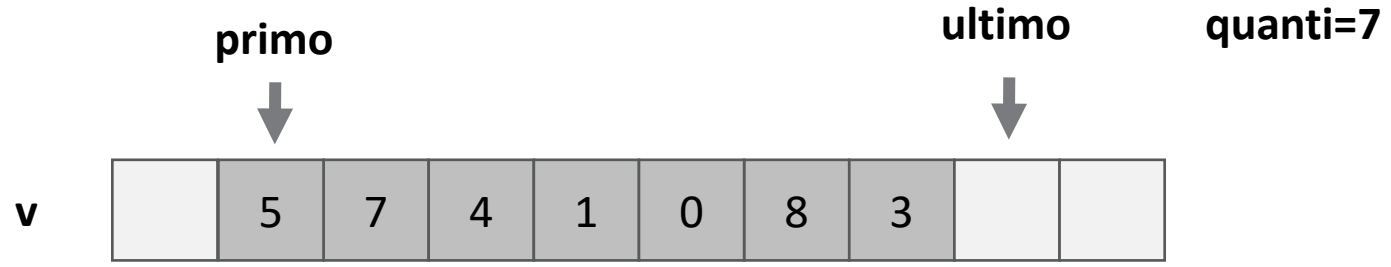
```
bool Coda::inserisci(int valore) {
    if (piena())
        return false;
    v[ultimo] = valore;
    ultimo = (ultimo+1)%DIM;
    return true;
}

bool Coda::estrai(int& valore) {
    if (vuota())
        return false;
    valore = v[primo];
    primo = (primo+1)%DIM;
    return true;
}
```

Classe: Coda (implementazione alternativa)

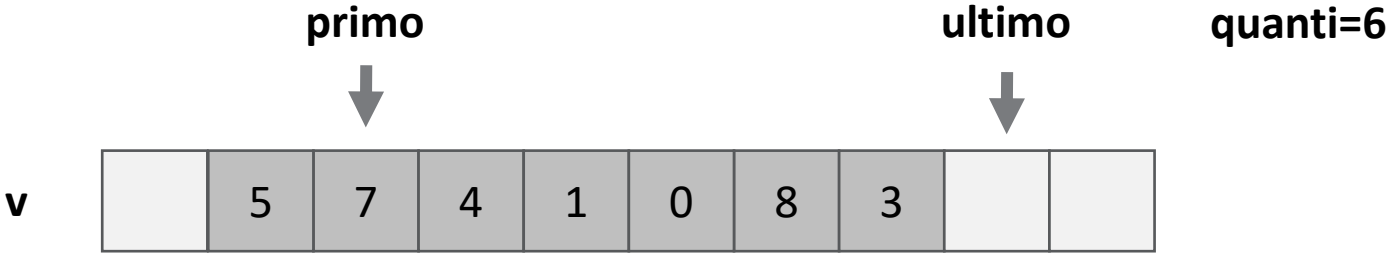
```
#include <iostream>
using namespace std;

const int DIM = 10;

class Coda {

    int v[DIM];
    int primo, ultimo;
    int quanti;

public:

    Coda();
    bool inserisci(int valore);
    bool estrai(int& valore);
    bool vuota();
    bool piena();
};
```



Inserimento:



Estrazione:



Classe: Coda (implementazione alternativa)

```
Coda::Coda() {
    primo = 0; ultimo = 0;
    quanti = 0;
}

bool Coda::vuota() {
    if (quanti == 0)
        return true;
    return false;
}

bool Coda::piena() {
    if (quanti == DIM)
        return true;
    return false;
}
```

```
bool Coda::inserisci(int valore) {
    if (piena())
        return false;
    v[ultimo] = valore;
    ultimo = (ultimo+1)%DIM;
    quanti++;
    return true;
}

bool Coda::estrai(int& valore) {
    if (vuota())
        return false;
    valore = v[primo];
    primo = (primo+1)%DIM;
    quanti--;
    return true;
}
```

Esercizio: Lista di iscrizione a una gara

- Si vuole realizzare una classe che memorizza la lista degli atleti iscritti a una gara podistica (max 10), in cui ogni atleta è identificato dal proprio cognome (tipo **string**)

- La classe deve mantenere il cognome di tutti gli atleti iscritti
  - Un insieme di informazioni dello stesso tipo (posso usare un **array**)

- Operazioni fondamentali:
  - Inizializzazione della lista
  - Inserimento di un atleta nella lista
  - Cancellazione di un atleta dalla lista, specificandone il cognome

Esercizio: Lista di iscrizione a una gara

```
const int MAX = 10;
class Iscrizione {

    string atleti[MAX];
    int numeroAtleti;

public:

    Iscrizione();
    void visualizzaLista();
    bool iscrivi(string nome);
    bool cancella(string nome);
};
```

Vettore di MAX elementi di tipo string

Esercizio: Lista di iscrizione a una gara

```
Iscrizione :: Iscrizione() {
    numeroAtleti = 0;
}

void Iscrizione ::visualizzaLista() {
    cout << "Lista:" << endl;
    for (int i=0; i<numeroAtleti; i++)
        cout << atleti[i] << endl;
}
```

Esercizio: Lista di iscrizione a una gara

```
bool Iscrizione::iscrivi(string nome) {
    if (numeroAtleti >= MAX)
        return false;
    else {
        atleti[numeroAtleti] = nome;
        numeroAtleti++;
    }
    return true;
}
bool Iscrizione::cancella(string nome) {
    for (int i=0; i<numeroAtleti; i++) {
        if (atleti[i] == nome) {
            atleti[i] = atleti[numeroAtleti-1];
            numeroAtleti--;
            return true;
        }
    }
    return false;
}
```

Esercizio: Lista di iscrizione a una gara

```
int main() {
  Iscrizione lista;
  string nuovoAtleta;
  cin >> nuovoAtleta;
  while (nuovoAtleta != "stop") {
      if ( !lista.iscrivi(nuovoAtleta) ) {
          cout << "Lista piena!" << endl;
          break;
      }
      cin >> nuovoAtleta;
  }
  lista.visualizzaLista();

  lista.cancella("Rossi");
  lista.cancella("Neri");
  l.visualizzaLista();

  return 0;
}
```

Esercizio: Lista di iscrizione a una gara

```
// iscrizione senza omonimi
bool Iscrizione ::iscrivi(string nome) {
    if (numeroAtleti >= MAX)
        return false;
    for (int i=0; i<numeroAtleti; i++) {
        if (nome == atleti[i])
            return false;
    }
    atleti[numeroAtleti] = nome;
    numeroAtleti++;
    return true;
}
```