ESERCIZI

- Si vuole realizzare un programma che consente di definire frazioni e di svolgere le quattro operazioni aritmetiche (+,-,*,/) tra due frazioni. Si vuole poi poter stampare le frazioni come rapporto num/den.
- Es.

```
Frazione a(3,2); // a = 3/2
Frazione b(3,8); // b = 3/8
a.somma(b); // a= a+b=30/16
cout << "a vale " << a.stampa(); // a vale
30/16
```

LISTA INIZIALIZZAZIONE

- Consideriamo la classe:

```
class X{
    const int a;
public:
    int b;
    X();
};
```

- Come si può inizializzare la variabile a?

```
class X{
    const int a = 4;
```

  darebbe errore poiché la dichiarazione non può allocare memoria

- come anche darebbe errore

```
X::X(int n){a=n; //a costante}
```

- Liste di inizializzazione: consentono di inizializzare (anziché assegnare)

  X::X(int **n**, int **m**): **a(n), b(m)** {}

- Subito dopo la lista dei parametri e prima del corpo della funzione, preceduta dai «:»

COSTRUTTORE DI COPIA DI DEFAULT

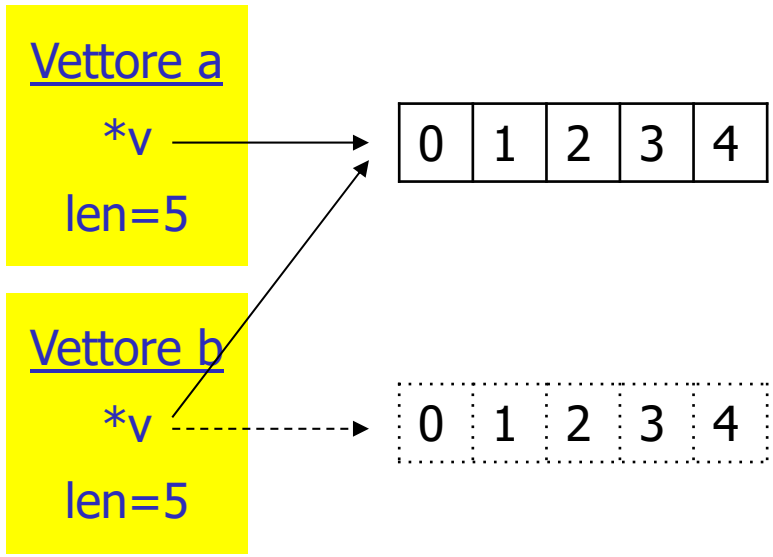
- Viene creato automaticamente dal compilatore
- Serve a creare un oggetto a partire da un altro oggetto
- Invocato quando:
  - Si inizializza un oggetto a partire da un altro oggetto
    - Es. `Persona p1;`

      …

      `Persona p2 = p1;` // Anche Persona p2(p1);
  - Un Oggetto è passato come argomento di ingresso per valore ad una funzione
  - Una funzione ritorna un Oggetto.
- Effettua una copia bit a bit dell'oggetto

LIMITI DEL COSTRUTTORE DI COPIA DI DEFAULT


```
classe Vettore(){
    int* v;
 public:
    int len;
    Vettore(int n){
      v= new int[n]; len=n;
    }
    void inserisci(i,x);
    // inserisce l'elemento
    // x alla posizione i
};
int main(){
    Vettore a(5);
    for(int i=0;i<5;i++)
          a.inserisci(i,i);
    Vettore b=a;//Vettore b(a);
}
```



COSTRUTTORE DI COPIA


```
classe Vettore(){
      int* v;
   public:
      int len;
      …
      Vettore(const Vettore& vett);
      …
};
Vettore::Vettore(const Vettore& vett){
      len=vett.len;
      v = new int[len];
      for(int i=0; i < len; i++)
            v[i]=vett.v[i];
}
```

DISTRUTTORE

- Anche il distruttore ha lo stesso nome della classe ma preceduto dal simbolo ~
- Non ha valore di ritorno e non può avere argomenti in ingresso
- E' unico
- Tipicamente usato per rilasciare (deallocare) memoria dinamicamente allocata nell'oggetto
- Se non dichiarato esplicitamente il compilatore ne crea uno vuoto
- Es.

```
class Persona {
   public:
    ~Persona(){cout<<"Ciao";}
};
```

ESEMPIO USO DISTRUTTORI


```
classe Vettore(){
   int len;
   int* v;
 public:
   Vettore(int n){
     v= new int[n];
   }
   ~Vettore(){
     delete [] v;
   }
};
```

```
int main{
   if(cond){
    Vettore vett(10);
     // operazioni
     // su vett
   }else{
     …
   }
}
```

ESEMPIO USO DISTRUTTORI


```
classe Vettore(){
   int len;
   int* v;
 public:
   Vettore(int n){
      v= new
   }
   ~Vettore(){
      delete [] v;
   }
};
```

Costruttore

Distruttore

```
int main{
   if(cond){
    Vettore vett(10);
      // operazioni
      // su vett
   }else{
      …
   }
}
```

ESERCIZI

- Creare un programma che consenta di gestire un torneo di beach-volley. Il torneo può coinvolgere un massimo di 10 squadre. E' possibile iscrivere una squadra al torneo, stampare l'elenco delle squadre e generare il calendario delle partite.
- Una squadra di beach-volley ha un nome ed è composta da 2 giocatori e 1 riserva. Per ogni giocatore bisogna ricordare nome, cognome, sesso (m/f), anno di nascita. Una squadra può essere inizialmente creata dandole un nome che non può essere poi modificato. Successivamente si possono aggiungere i giocatori specificando se un giocatore è titolare o panchina.
- Per una squadra è possibile stampare l'elenco dei giocatori ed in particolare nome, cognome, anno, ruolo del giocatore (titolare/panchina).