Si risolvano i due esercizi proposti. Scegliere liberamente se separare gli *header* dalle implementazioni oppure se includere tutto in due file `.cpp` (uno per esercizio). Si consiglia di concentrarsi sulla pulizia del codice: pochi punti risolti in modo ordinato saranno valutati meglio di tanti punti trattati disordinatamente. La sufficienza è raggiunta risolvendo correttamente i primi due punti del primo esercizio oppure i primi due del secondo. L'ultimo punto di ciascun esercizo è piú avanzato, si consiglia di affrontarlo solo se si è terminato il resto.

Esercizio 1

Si vuole scrivere una classe `Rational` che descriva numeri razionali non negativi.

1. Si pongano come membri privati due variabili `unsigned int`, corrispondenti al numeratore e al denominatore. Tra i membri pubblici, si scriva un opportuno costruttore che richieda come parametri il numeratore e il denominatore, con valori di default rispettivamente 0 e 1. Si implementino inoltre due *access function* per leggere le due variabili private.

2. Si scrivano il costruttore di copie e l'operatore di assegnazione. Inoltre si scrivano l'operatore "+" e l'operatore "∗", che restituiscano rispettivamente la somma e il prodotto di due frazioni, non necessariamente ridotte ai minimi termini.

3. Si implementi l'overloading dell'operatore

   ```
   ostream & operator<<
   (ostream & outstream, const Rational & r).
   ```

   Esso dovrà stampare il generico numero razionale nella forma "n/d", dove n è il numeratore e d il denominatore. Si implementi un controllo su d in modo che i numeri interi vengano stampati nella forma standard.

4. Si verifichi il comportamento del codice con un `main` che esegua le seguenti istruzioni:

   - istanziare due numeri razionali, inizializzandoli a 2/3 e 5/9 rispettivamente;

- istanziare un numero razionale, inizializzandolo con la somma dei due precedenti (in modo che venga richiamato il costruttore di copie);
- istanziare un numero razionale e uguagliarlo poi al prodotto dei primi due (in modo che venga richiamato l'operatore di assegnazione);
- stampare su `std::cout` i quattro valori.

5. Si implementi, tra i membri privati della classe `Rational`, una funzione `MCD` che calcoli il massimo comune divisore tra il numeratore e il denominatore. Si implementi poi una funzione `simplify` tra i membri pubblici, che semplifichi la frazione ai minimi termini, aiutandosi con `MCD`.

Esercizio 2

Si vuole scrivere del codice per gestire i tempi di percorrenza dei treni tra una stazione e l'altra.

1. Si scriva una classe `Station`, che abbia come membri privati due numeri reali corrispondenti alle coordinate cartesiane della stazione (in km). Si implementino, nella parte pubblica, un opportuno costruttore che fissi le coordinate e il costruttore di copie. (Non è necessario il costruttore di default).

2. Si dichiari, all'interno della classe `Station`, una funzione `friend` globale chiamata `dist`, che prenda due referenze a oggetti di tipo `Station` e restituisca la distanza tra le due stazioni (semplicemente la distanza in linea d'aria, col teorema di Pitagora). Si implementi la funzione `dist` e si verifichi il comportamento del codice con il seguente `main`:

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

int main() {
  Station Milano(0.,0.);
  Station Bergamo(35.,25.);
  Station Piacenza(40.,-45.);
```

```
  cout << dist(Milano, Bergamo) << endl;
       << dist(Bergamo, Piacenza) << endl;
       << dist(Piacenza, Milano) << endl;
}
```

3. Si scriva una classe `Train`, che abbia come membro privato un numero reale corrispondente alla velocità media del treno (in km/h). Nella parte pubblica si implementino le seguenti funzioni: un opportuno costruttore che fissi la velocità; una *access function* che permetta di leggerla; una funzione `time_of_travel` che prenda due referenze a oggetti di tipo `Station` e restituisca il tempo in ore necessario a coprire la distanza tra le due stazioni (per risolvere il punto 5, definire questa funzione come virtuale).

4. Si scrivano due classi `Regionale` e `Intercity` che ereditino pubblicamente da `Train`. Il treno regionale ha una velocità media fissata di 60 km/h (da implementare nel costruttore) e compie il tragitto più breve tra una stazione e l'altra (dunque il metodo `time_of_travel` già implementato per la classe base non ha bisogno di essere riscritto). Il treno intercity invece ha una velocità media di 110 km/h e possiede, tra i suoi membri privati, un oggetto di tipo `Station`, che rappresenta uno scalo obbligato per ogni tragitto (da inizializzare in un costruttore che prenda un opportuno parametro). Il calcolo del tempo di percorrenza tra due stazioni, in questo caso, sarà quindi la somma del tempo di percorrenza tra la stazione di partenza e quella di scalo, e tra la stazione di scalo e quella finale.

5. Per verificare il comportamento polimorfico, si scriva un `main` in cui si istanzia un `vector` di puntatori a oggetti di tipo `Train` e lo si riempie con due oggetti: un `Regionale` e un `Intercity`, quest'ultimo con stazione di scalo Milano. Si stampi a schermo per ogni elemento del vettore il tempo di percorrenza tra Bergamo e Piacenza.