Si risolva l'esercizio proposto. Per facilitare la correzione, se possibile includere tutto in un unico file sorgente. La sufficienza è raggiunta risolvendo correttamente i primi tre punti.

La sequenza di Thue-Morse

1. Si scriva una classe `Sequence` che rappresenti una sequenza di 16 numeri interi. Tra i membri *protected* si metta un `std::vector` di interi, che conterrà gli elementi della sequenza. Tra i membri *public* si scriva un costruttore che richieda tre parametri interi a, b, c (con valori di default nulli) e generi la sequenza $ai + bi^2 + ci^3$, con $i = 1, 2, \ldots, 16$.

2. Si scriva il costruttore di copie ed una funzione `print` che stampi su `cout` la sequenza.

3. Si implementi un overloading dell'operatore "`*`" (membro della classe) che calcoli il prodotto scalare tra due sequenze. Il prodotto scalare tra $\{a_1, a_2, \ldots, a_{16}\}$ e $\{b_1, b_2, \ldots, b_{16}\}$ è definito come $a_1b_1 + a_2b_2 + \cdots + a_{16}b_{16}$.

4. Si scriva un `main` che istanzi due sequenze con parametri a, b, c diversi, ad esempio $(1, 0, 0)$ e $(0, 0, 1)$. Si controlli, stampando il risultato su `cout`, che il prodotto tra loro sia commutativo.

5. Si scriva una classe `BinarySequence` che erediti pubblicamente da `Sequence`: essa rappresenterà sequenze composte solo da 0 e 1. Il costruttore di default (senza parametri) dovrà generare una sequenza di zeri e uni scelti in modo casuale.

6. Tra i membri pubblici, si implementi una funzione `not` (la negazione logica) che restituisca una `BinarySequence` con uni e zeri invertiti. Si implementi inoltre una funzione membro `cost`, che prenda come parametro una referenza ad un oggetto di tipo `Sequence` s, e restituisca un numero intero k, calcolato come segue. Chiamiamo b la sequenza binaria (l'oggetto che fa la chiamata a `cost`); allora k è la differenza tra il prodotto scalare di s con b e il prodotto scalare di s con la negazione di b. La quantità k è chiamata costo di b su s.

7. Si implementi un costruttore di `BinarySequence` che prenda come parametro un numero intero positivo e costruisca la sua rappresentazione binaria [può tornare utile la funzione `pow(double base,double esponente)` della libreria `<cmath>`, che restituisce la potenza $\text{base}^{\text{esponente}}$].

8. Nel `main` si riempia un `std::vector` di `BinarySequence` corrispondenti alle rappresentazioni binarie di tutti gli interi tra 1 e $2^{16} - 1$. Tra queste, se ne vuole trovare una (chiamiamola t) con la proprietà che il costo di t su

qualunque sequenza di quelle generate dal costruttore del punto (1) sia nullo (una condizione equivalente è che sia nullo il costo sulle tre sequenze i, i^2 e i^3). Per fare questo, si scriva un *predicato* (una funzione unaria globale che restituisca un `bool`), che identifichi se una sequenza ha questa proprietà. Quindi, per trovare la sequenza cercata tra quelle nel `std::vector`, si utilizzi l'algoritmo

```
std::find_if(it1, it2, pred)
```

che scorre gli elementi tra l'iteratore it1 (compreso) e l'iteratore it2 (escluso) e resituisce un iteratore al primo elemento per cui il predicato `pred` è vero. Si stampi la sequenza su `cout`.