

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 quattro punti.

## Esercizio

Si vuole scrivere un semplice framework per gestire codici a barre (rappresentati come sequenze di cifre).

1. Si scriva una classe `Digit`, che rappresenterà una cifra del codice. Questa è rappresentata da un valore intero  $v$  sottostante e da un modulo  $m$  intero: la cifra sarà quindi uguale a  $v \bmod m$ . Si mettano  $m$  e  $v$  tra i membri privati e si scriva un opportuno costruttore che li inizializzi, con *valori di default*  $m = 8$  e  $v = 0$ . Attenzione: si vuole che il costruttore, se chiamato con un solo parametro, imposti  $v$  e assegni il valore di default al modulo  $m$ .
2. Si scriva il costruttore di copie di `Digit` (usando la sintassi della lista di inizializzazione) e due *access function*. Queste ultime dovranno permettere esclusivamente di scrivere  $v$  e di leggere la cifra (come spiegato al punto 1).
3. Si scriva l'operatore "`==`", che confronta due cifre (contano le *cifre*, a prescindere dai valori sottostanti o dai loro moduli). Si abbia cura di *riutilizzare* il codice già scritto al punto 2, tramite chiamata a funzione.
4. Si scriva una classe `Barcode`, che rappresenterà una sequenza di cifre. Tra i membri privati si metta quindi un `std::vector` di `Digit`. Si scriva un costruttore che prenda come argomenti due iteratori `it1` e `it2` opportuni, e riempia il vettore privato copiando gli elementi compresi tra `it1` incluso e `it2` escluso. Bonus: si provi a scrivere quest'ultima operazione senza usare cicli.
5. Si scrivano, tra i membri pubblici, una funzione `codeLength()` che restituisca la lunghezza della sequenza, e una funzione `print()` che stampi su `cout` le cifre in sequenza. Si vuole che la funzione `print()` abbia comportamento polimorfico.
6. Tra i membri protetti, scrivere una *access function* che restituisca (per copia) il vettore di cifre. Rispondere brevemente (in un commento): dall'esterno della classe `Barcode` è possibile usare questa *access function* per modificare il vettore?
7. Si scriva una classe `CBarcode`, derivata di `Barcode`, che implementerà in più un meccanismo di *checksum*, cioè un valore di controllo calcolato deterministicamente a partire dalle cifre della sequenza. In particolare, il valore di controllo è calcolato come la somma delle cifre, modulo 16. Mettere tra i membri privati

una variabile `checksum` e scrivere un costruttore che, oltre ad implementare la stessa funzionalità di quello di `Barcode`, la calcoli.

8. Si scriva l'*override* della funzione `print()`. Il comportamento sarà quello di stampare la sequenza di cifre, preceduta dal valore di `checksum` tra parentesi tonde. Si riutilizzi il codice già scritto per la classe base, attraverso una chiamata a funzione.
9. Per controllare il comportamento polimorfico, nel *main* si istanzi un `std::vector` che contenga un `Barcode` e un `CBarcode` allocati *dinamicamente*; si consideri se usare valori, puntatori o referenze. Si chiami poi la funzione `print()` di ogni elemento del vettore. Avendo bisogno di istanziare un vettore di `Digit`, si usi la seguente linea di codice:

```
std::vector<Digit> digits = {4,1,2,10,9}.
```

(Ci si ricordi di deallocare gli oggetti allocati dinamicamente.)

10. Si scriva una versione *generica* del costruttore di `Barcode`, che accetti come argomenti due iteratori a una struttura qualsiasi. Si controlli la funzionalità nel *main* inizializzando un nuovo `Barcode` usando iteratori a elementi di un container `std::list`.
11. Se ora si volesse istanziare un `std::set<Barcode>` e inserirvi degli oggetti si andrebbe incontro a un errore, dovuto alla mancanza di una funzionalità. Quale? In quale modo la si potrebbe implementare? Commentare brevemente senza necessariamente scrivere codice.