

## Esercizio 1

Data la seguente porzione di programma rispondere alle domande corrispondenti:

```
void f (int& a) {
    int& b = a;
    ++a;
    if (a != b) { a = 7; }
    else { a = 0; }
}

int main() {
    int* matricola = new int[6]  { //Inserire qui la tua matricola };

    // 1. Quanto vale matricola[0] dopo l'invocazione della funzione f dichiarata sopra?

    f(matricola[0]);

    // 2. Perché la seguente istruzione non è corretta?

    matricola.push_back(9);

    // 3. Quale delle seguenti operazioni di stampa (a. b. o c.) è corretta? Perché?

    int v = 5;
    int* q = &v;

    // a. cout << matricola[*q] << endl;
    // b. cout << matricola[q] << endl;
    // c. Nessuna delle due;

    // 4. Indicare tutte le istruzioni da eseguire per deallocare la memoria dinamica
    //    allocata durante l'esecuzione del programma.

    // a. delete[] matricola;
    // b. delete[] q;
    // c. delete matricola;
    // d. delete q;
    // e. for (int i=0; i < 6; i++) delete matricola[i];
    // f. nessuna delle precedenti

    return 0;
}
```

## Esercizio 2

Si richiede di implementare la classe `Videogioco` secondo le seguenti linee guida: un videogioco è caratterizzato da: un titolo, un anno di pubblicazione, un prezzo, un genere, un insieme di console per cui è disponibile. Deve poter essere possibile leggere tutti i campi dall'esterno della classe ma non di modificarli, ad eccezione del prezzo, che può essere modificato solo "in diminuzione" (se provo a modificare il prezzo del videogioco aumentandolo, l'operazione non deve avere effetto). *Si può, se si ritiene necessario, arricchire la classe `Videogioco` con ulteriori metodi/funzionalità, purché non vadano in contrasto con le richieste precedenti.*

Vogliamo inoltre progettare una classe `CatalogoVideogiochi`, che conservi il catalogo di videogiochi disponibili per l'acquisto in un negozio. Un `CatalogoVideogiochi` deve far riferimento ad una determinata console (memorizzata a tempo di costruzione dell'oggetto e non più modificabile) e conserva un insieme di `videogiochi` ordinati per anno dal più recente al più

vecchio, e, a parità di anno di pubblicazione, per prezzo decrescente (dal più caro al più economico). CatalogoVideogiochi deve garantire almeno le seguenti funzionalità:

- **Aggiungere un videogiochi al catalogo:** Il videogiochi deve essere aggiunto solo se è disponibile per la console a cui il catalogo fa riferimento. In caso contrario l'operazione di aggiunta non ha alcun effetto. Dopo l'aggiunta, il catalogo deve rimanere ordinato secondo i criteri descritti prima.
- **Consigliare un videogiochi:** questo metodo riceve un parametro (numero intero positivo) `budget` e restituisce il videogiochi più recente che abbia un prezzo minore del parametro `budget`. Se non esiste alcun videogiochi che soddisfa i criteri, il programma deve restituire un videogiochi con titolo "ERRORE".

Per l'implementazione delle classi si può far uso di strutture e metodi della libreria standard di c++. Si richiede di usare **const** ove opportuno.

### Esercizio 3

Data un'istanza di `AlberoB<int> tree` e un `vector<int> s`, scrivere una funzione `esercizio3` che restituisca un `vector<int>` contenente tutti gli elementi di `s` che compaiono in `tree` un numero dispari di volte.

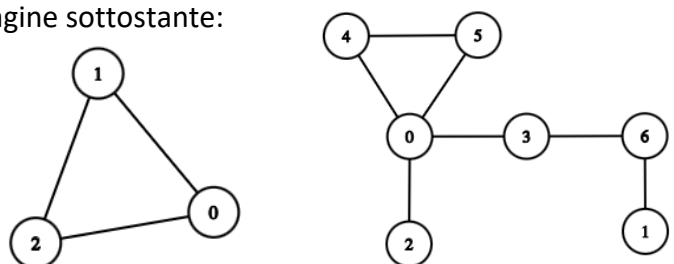
La classe `AlberoB<T>` mette a disposizione la seguente interfaccia pubblica (`tree` istanza di `AlberoB<T>`):

- `tree.radice()` restituisce il valore informativo di `tree` (di tipo `T`)
- `tree.figlio(DIR)` restituisce il sottoalbero sinistro (`DIR=SIN`) e destro (`DIR=DES`) di `tree`
- `tree.nullo()` restituisce `true` se `tree` è un albero nullo e `false` altrimenti
- `tree.foglia()` restituisce `true` se `tree` è una foglia e `false` altrimenti

### Esercizio 4

Dati due grafi `G1` e `G2`, diciamo che `G1` è **isomorfo** ad un sottografo di `G2` se è possibile associare **tutti** i nodi di `G1` ad alcuni dei nodi di `G2` in modo tale che `G1` sia sovrapponibile alla porzione di nodi considerata su `G2`, come si può notare dall'immagine sottostante:

In questo esempio, il grafo sulla sinistra risulta isomorfo ad una porzione del grafo sulla destra in quanto i nodi **0,1,2** del primo possono essere "sovrapposti" rispettivamente ai nodi **0,4,5** del secondo. Per verificare questa condizione basta controllare che tutti gli archi che esistono tra 0,1,2 nel primo grafo esistono anche tra i nodi 0,4,5 del secondo grafo



Scrivere un programma C++ che, presi in input due Grafi `G` e `H` (rappresentati mediante la classe `Grafo`), restituisca `true` se e solo se `G` è **isomorfo** ad un qualsiasi sottografo di `H`.

La classe `Grafo` mette a disposizione la seguente interfaccia di metodi costanti (con `g` istanza della classe `grafa` e `i` e `j` nodi di `g`):

- Il metodo `g.n()` restituisce il numero di nodi di `g`
- Il metodo `g.m()` restituisce il numero di archi di `g`
- Il metodo `g(i, j)` restituisce `true` se esiste un arco dal nodo `i` al nodo `j` e `false` altrimenti