UNIVERSITÀ DELLA CALABRIA

Esercizio 1

Data la seguente porzione di programma rispondere, dando motivazione, alle domande corrispondenti:

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

    // 1. La seguente istruzione è corretta? Se si, cosa stampa?

    cout << matricola + 4 << endl;

    // 2. La seguente istruzione è corretta? Se si, che cosa stampa?

    *(matricola + 3)+=2;
    cout << *(matricola)+3 << " " << *(matricola+3) << endl;

    // 3. Le seguenti istruzioni sono corrette? Motivare la risposta

    int * nuova_matricola = new int[3];

    for (int i=0; i < 6; i+=2) {
       nuova_matricola[i/2] = matricola[i] + matricola[i+1];
    }

    // 4. Scrivere sul foglio le istruzioni per deallocare la memoria
    dinamica allocata nel programma. Scrivere "NIENTE" se non c'è bisogno di
    deallocare memoria

    return 0;

}
```

Esercizio 2

Un nostro amico, che di professione fa il *food blogger*, ci ha chiesto di realizzare un software in grado di tenere traccia delle sue recensioni ai ristoranti che visita. Per ogni locale c'è bisogno di conservarne alcune caratteristiche in una classe. In particolare, siamo interessati a conoscerne il nome (`string`), la località (`string`), le coordinate geografiche (`pair<double,double>`), a cui è abbinata una recensione, composta da un voto che va da 1 a 5 per le categorie: "prezzo", "qualità", "location", "originalità". Si deve assumere che un ristorante `r` sia identificato univocamente dalla coppia nome+località.
Dobbiamo quindi progettare ed implementare una classe `OrganizzaRecensioni`, che metta a disposizione *almeno* le seguenti funzionalità:

- Aggiungere una nuova recensione per un ristorante, ricevendo in input tutto il necessario. Nel caso in cui lo stesso ristorante abbia già ricevuto una recensione in passato, la nuova recensione dovrà sovrascrivere la precedente;
- Dato in input un ristorante ed una categoria, restituire la corrispondente valutazione;
- Dati in input un intero `x` e due stringhe `a` e `b`, restituire tutti i ristoranti che abbiano un punteggio uguale o superiore a `x` nella categoria `a`, ordinate in modo decrescente in base al relativo punteggio sulla categoria `b`.

UNIVERSITÀ DELLA CALABRIA

Esercizio 3

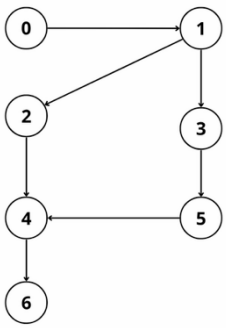
Scrivere una funzione che, prendendo in input un grafo orientato `g` (implementato tramite la classe `Grafo`) e tre suoi nodi, indicati rispettivamente con `c,b,v`, che restituisca il percorso più breve (`vector<unsigned>`) per arrivare da `c` a `b` passando per `v`. Se non è possibile trovarlo (ad esempio, da `v` non è possibile raggiungere `b`), allora, il programma deve restituire un `vector<unsigned>` di dimensione 0. È possibile che alcuni nodi del grafo si ripetano nella prima (da `c` a `v`) e nella seconda (da `b` a `v`) parte del cammino trovato.

La classe `Grafo` mette a disposizione la seguente interfaccia pubblica di metodi costanti (con `g` istanza della classe):

- `g.n()` restituisce il numero di nodi del grafo,
- `g.m()` restituisce il numero di archi del grafo,
- `g(i,j)` restituisce `true` se esiste l'arco diretto tra il nodo `i` e il nodo `j`.

I nodi sono etichettati con i valori da `0` a `g.n()-1`.

Esempio: per il grafo in figura, con c=0, v=5 e b=6, il programma deve restituire {0,1,3,5,4,6}, mentre, se scegliamo c=0, v=2 e b=5, il programma restituirà un vector vuoto {}



Esercizio 4

Scrivere una funzione `esercizio4` che prenda in input una `struct` contenente:

- un `vector<string> studenti` (si può supporre che non contenga duplicati)
- un `vector pair<string,string> inc` in cui ogni `pair` rappresenta l'incompatibilità dei due studenti nel condividere la stanza
- un `vector<pair<unsigned,vector<string>>> stanze`, in cui il <u>first</u> (`unsigned`) identifica univocamente una <u>stanza</u>, mentre il rispettivo <u>second</u> (`vector<string>`) tiene traccia dei suoi <u>occupanti</u>.

L'obiettivo è quello di assegnare ad <u>ogni</u> studente un posto letto all'interno delle stanze del dormitorio, secondo i criteri di seguito:

- Non è possibile creare nuove stanze al di fuori di quelle date in input (si può assumere che inizialmente tutte le stanze siano vuote, cioè senza occupanti)
- La capienza di ogni stanza è data dal resto della divisione tra il numero identificativo della stanza e `6` incrementato di `1`. Non è ammesso che una stanza abbia un numero di occupanti superiore alla sua capienza.
- <u>Tutti</u> i vincoli di incompatibilità espressi dagli studenti devono essere rispettati,
- Nessuna stanza deve restare vuota (*È possibile che qualche stanza non sia completamente riempita, ma non è possibile lasciare stanze completamente vuote),*
- Tutti gli studenti devono avere una sistemazione.

La funzione deve ritornare `true` se e solo se è possibile sistemare tutti gli studenti in modo tale che siano rispettate le seguenti elencate prima, e `false` altrimenti.