Fondamenti di Programmazione 2
(Corso di Laurea in Informatica)
Esame del 14/9/2023

UNIVERSITÀ DELLA CALABRIA

Esercizio 1

Data la seguente porzione di programma rispondere alle domande corrispondenti:

```
int main() {
  // Scrivi sul foglio la tua matricola
  int* matricola = new int[6] {...};

  // 1. La seguente istruzione è corretta? Se sì, cosa stampa?
  cout << *(matricola + 1) * (*(matricola + 2)) << endl;

  // 2. La seguente istruzione è corretta? Se sì, cosa stampa?
  cout << *(*matricola + 3) << endl;

  // 3. Cosa stampa la seguente porzione di codice?
  int& z = matricola[5];
  for (int i = 0; i < 6; ++i) {
    matricola[i] = z;
  }
  z = 17;
  for (int i = 0; i < 6; ++i) {
    cout << matricola[i];
  }
  cout << endl;

  // 4. Come andrebbe deallocata la memoria dinamica allocata inizialmente?
  // A: for (int i = 0; i < 6; ++i) delete matricola[i];
  // B: delete matricola;
  // C: Sono necessarie entrambe le operazioni A, B.
  // D: Tutte le risposte precedenti sono errate.

  return 0;
}
```

Esercizio 2

Implementare una classe Esercizio che contenga (almeno) i campi privati descrizione e difficoltà rispettivamente di tipo std::string e int.
Successivamente, implementare una classe BancaDatiEsercizi, per la gestione di un insieme di esercizi. La soluzione proposta dovrà implementare (almeno) i seguenti metodi:

- bool aggiungiEsercizio(std::string id, Esercizio e)
  - Aggiunge un esercizio alla banca dati, identificato univocamente da una stringa (id). Non è possibile aggiungere alla banca dati due esercizi con lo stesso id. Il metodo restituisce true se l'esercizio è stato aggiunto correttamente, altrimenti false.
- vector<Esercizio> creaEsame(int n, int diff_M, int rep_M);
  - Facendo in modo che la classe BancaDatiEsercizi tenga traccia di quante volte un esercizio sia stato già assegnato in precedenza, la funzione crea un esame selezionando n esercizi dalla banca dati che abbiano una difficoltà totale (ottenuta sommando le difficoltà dei singoli esercizi) minore o uguale a diff_M, selezionando gli esercizi tra quelli che sono stati già assegnati meno di rep_M volte. Se non fosse possibile creare un esame con tali criteri, il vector restituito sarà vuoto.

Nella segnatura di tutti i metodi di cui sopra, utilizzare const ovunque sia opportuno.

UNIVERSITÀ DELLA CALABRIA

Esercizio 3

Scrivere una funzione **esercizio3** che presi in input un grafo orientato G e due suoi vertici u, v restituisca l'insieme dei nodi x tale che $\delta(u, x) = \delta(x, v)$, dove $\delta(a, b)$ denota la lunghezza del cammino minimo tra i nodi a e b nel grafo G.

La funzione dovrà avere la seguente segnatura:

std::vector<int> esercizio3(const Grafo& g, int u, int v)

Il grafo è rappresentato da una classe Grafo con la seguente interfaccia (con g un'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 numeri interi da 0 a g.n()-1.

Esercizio 4

Siano $x_1, \ldots, x_n$ delle variabili binarie, cioè che possono assumere come valore *vero* (V) o *falso* (F). Una terna (x_i, x_j, x_k), con $0 \leq i < j < k < n$ si dice *soddisfatta* quando **ad esattamente una** delle variabili x_i, x_j, x_k è assegnato il valore V, altrimenti la terna si dice *non soddisfatta*.

Scrivere una funzione **esercizio4** che presi in input un insieme di m terne di variabili e il numero n di variabili disponibili, restituisca un possibile insieme di variabili cui deve essere assegnato il valore V affinché ciascuna terna risulti soddisfatta.

- La terna (x_i, x_j, x_k) è rappresentata da un std::vector<int> contenente gli interi {i, j, k};
- L'insieme di terne è rappresentato da un std::vector<std::vector<int>>;
- Si può assumere l'input sia ben formato, cioè tutte le variabili che appaiono nelle terne sono della forma x_i per $0 < i < n + 1$ e nessuna terna contiene due volte la stessa variabile;

Esempio. Consideriamo $n = 6$ variabili e le seguenti terne: [1,2,3], [2,3,4], [3,4,5], [4,5,6], [5,6,1].

Un esempio di assegnamento che soddisfa tutte le terne è:
V = {3,6}, F = {1,2,4,5}
[1,2,**3**], [2,**3**,4], [**3**,4,5], [4,5,**6**], [5,**6**,1]

Un esempio di assegnamento non valido è:
V = {4}, F = {1,2,3,5,6}
[1,2,3], [2,3,**4**], [3,**4**,5], [**4**,5,6], [5,6,1] - Le terne [1,2,3] e [5,6,1] non sono soddisfatte;

V = {4,1,2}, F = {3,5,6}
[**1**,**2**,3], [**2**,3,**4**], [3,**4**,5], [**4**,5,6], [5,6,**1**] - Le terne [1,2,3] e [2,3,4] hanno più di una variabile positiva.