Fondamenti di Programmazione 2
(Corso di Laurea in Informatica)
Esame del 15/02/2024

UNIVERSITÀ DELLA CALABRIA

Esercizio 1

Data la seguente porzione di programma rispondere alle domande corrispondenti:

```
void g(int& a) { a = 42; }
int main() {
   //scrivi sul foglio la tua matricola
   int* matricola = new int[6]{..la tua matricola..};
   int* p = new int(matricola[3]);

   //1. Quale delle seguenti istruzioni è corretta e stampa il contenuto
     dell'array "matricola" (indicare una sola risposta)?
   for (int i=0 ; i < 6; i++) {
      //A: cout << *(matricola+i);
      //B: cout << *(matricola[i]);
      //C: cout << matricola+i;
      //D: nessuna delle precedenti;
   }

   //2. La seguente coppia di istruzioni è corretta? Se sì, cosa stampa?
   g(matricola[0]);
   cout << matricola[0] << endl;

   //3. La seguente istruzione è corretta? Se sì, cosa stampa?
   cout << p << endl;

   delete[] matricola;
   //4. Rispondi con vero/falso: A seguito dell'istruzione
   "delete[] matricola", è necessario deallocare anche "p"?

   return 0;
}
```

Esercizio 2

Implementare una classe **StackConInteressi** che modelli una pila di **double**. La classe dovrà implementare (almeno) i seguenti metodi:

- Un costruttore con parametro **t** che denota il **tasso di interesse** dello **StackConInteressi** (vedi metodo **pop**);
- Il costruttore per copia per la classe **StackConInteressi**
- Il distruttore per la classe **StackConInteressi**
- Un metodo **void push(double)** per inserire un **double** nella pila (N.B. non si può assumere che ci sia una dimensione massima per lo stack);
- Un metodo **double pop()** che restituisca il valore **t·e** dove **t** è il tasso di interesse della pila e **e** è l'elemento in cima alla pila. Questa operazione, inoltre, rimuoverà **e** dalla pila.
- L'operatore di assegnamento per la classe **StackConInteressi**

In particolare, **StackConInteresse** dovrà essere implementato utilizzando un array dinamico di **double**, e non è possibile utilizzare **std::stack**, **std::list**, **std::vector** o **std::queue** nella sua implementazione.
Segue un esempio di utilizzo della classe:

```
s = StackConInteressi(1.75);
s.push(8.0);
s.push(4.0);
cout << s.pop() << endl; // Stampa 7.0 = 4.0 * 1.75
```

UNIVERSITÀ DELLA CALABRIA

Esercizio 3

Scrivere una funzione che preso in input un'istanza di **AlberoB<char>** restituisca **true** se e solo se il contenuto di tutte le sue foglie, letto da sinistra a destra, rappresenta una sequenza di caratteri in ordine strettamente crescente.

NOTA: La classe template AlberoB<T> possiede la seguente interfaccia pubblica, dove t è un'istanza della classe:

- t.radice() - Restituisce il valore informativo della radice di t.
- t.foglia() - Restituisce true se t è una foglia
- t.nullo() - Restituisce true se t è l'albero vuoto, false altrimenti.
- t.figlio(d) - Restituisce il sottoalbero sinistro (se d == SIN) o destro (se d == DES).

Esercizio 4

L'università deve organizzare **m** corsi di recupero per **n** studenti. Non tutti gli studenti devono seguire tutti i corsi (per fortuna!), ma bisogna garantire che, per ciascuno studente, non vi siano collisioni di orario tra i vari corsi da seguire. In particolare, ad ogni corso deve essere assegnato un unico slot orario tra **k** possibili scelte disponibili. Si può assumere che ciascuno slot sia associato ad un numero compreso tra 1 e k e che più corsi possano essere allocati nello stesso slot purchè non violino il vincolo suesposto.

Scrivere una funzione che preso in input un **vector<vector<unsigned>>**, dove l'**i**-esimo vettore rappresenta l'insieme dei corsi che l'**i**-esimo studente deve seguire (ciascuno rappresentato da un intero positivo univoco; vedi esempio), stampi **a video** una possibile organizzazione dei corsi se esiste, altrimenti stampi **IMPOSSIBILE**. Se esistono più organizzazioni possibili si stamperà la prima soluzione individuata.

Esempio: Il vettore **[[1,2,3], [3,4], [1,5]]** rappresenta un'istanza del problema con **n=3** e **m=5**. In particolare, il primo studente dovrà seguire i corsi **(1,2,3)**, il secondo i corsi **(3,4)** e il terzo i corsi **(1,5)**. In questo caso, i corsi **3** e **4** non potranno essere organizzati nello stesso slot orario in quanto il secondo studente deve essere in grado di seguire entrambi, e analogamente i corsi **1** e **5** per il terzo studente ed i corsi **1,2,3** per il primo studente. *NB: **k** non dipende né dal numero di studenti né dal numero di corsi, ma è fornito in input!*

In questo esempio, se k=3, una possibile organizzazione dei corsi che rispetta i vincoli (non necessariamente l'unica) esiste e la funzione potrà stampare:

1,4

2,5

3

Indicante che i corsi 1 e 4 saranno assegnati al primo slot, i corsi 2 e 5 saranno assegnati al secondo slot ed il corso 3 sarà assegnato al terzo slot. N.B. potrebbero esistere soluzioni che non usano tutti gli slot disponibili, ma non si possono utilizzare più di k slot.