FONDAMENTI DI INFORMATICA I
CORSO DI LAUREA IN INGEGNERIA INFORMATICA
PROVA PRATICA 12 GENNAIO 2016

Un' `Aula` di personal computer è composta da un numero finito di postazioni. Le postazioni sono numerate a partire da 1. Ad ogni istante ciascuna postazione può essere libera oppure occupata da un utente. Ciascun utente è identificato attraverso una stringa. Le stringhe possono avere lunghezza qualsiasi e contenere qualunque carattere.

Implementare le seguenti operazioni che possono essere effettuate su di un'`Aula`:

--- PRIMA PARTE --- *(qualora siano presenti errori di compilazione, collegamento o esecuzione in questa parte, **l'intera prova sarà considerata insufficiente** e pertanto **non sarà corretta**)*

✓ **`Aula a(N);`**

Costruttore che crea un'`Aula` con `N` postazioni. All'inizio tutte le postazioni sono libere.

✓ **`a.aggiungi(id);`**

Operazione che aggiunge l'utente avente identificatore `id` all'aula `a`, in corrispondenza della prima postazione libera. Qualora l'aula sia piena, restituisce `false`, altrimenti `true`**. La aggiungi fallisce anche nel caso in cui si tenti di aggiungere un utente con `id` uguale a quello di uno già presente in aula**, nel qual caso restituisce ancora `false`.

✓ **`cout << a;`**

Operatore di uscita per il tipo `Aula`. L'uscita ha il seguente formato:

```
POSTAZIONE1:Utente1
POSTAZIONE2:<libera>
POSTAZIONE3:Utente3
POSTAZIONE4:<libera>
```

L'output mostrato corrisponde a una `Aula` di 4 postazioni, di cui la prima e la terza occupata da utente1 e utente3, rispettivamente. Le altre postazioni sono libere.

✓ **`a.elimina(p);`**

Operazione che libera la postazione `p`-esima.

--- SECONDA PARTE ---

✓ **`Aula a2(a);`**

Costruttore di copia, che costruisce `a2` utilizzando `a`.

✓ **`!a;`**

Operazione di negazione logica, che, **qualora l'aula sia piena**, sposta gli utenti fra le varie postazioni in modo che compaiano in ordine alfabetico. **Nel caso l'aula non sia piena non fa nulla.**

Esempio: Nel caso in cui `a` prima dell'ordinamento sia piena e con i seguenti utenti:

```
POSTAZIONE1:Utente1
POSTAZIONE2:Zenone
POSTAZIONE3:Utente3
POSTAZIONE4:Achille
```

dopo l'ordinamento `a` apparirà così:

```
POSTAZIONE1:Achille
POSTAZIONE2:Utente1
POSTAZIONE3:Utente3
POSTAZIONE4:Zenone
```

`~Aula();`
Distruttore.

Mediante il Linguaggio C++, realizzare il tipo di dato astratto **`Aula`**, definito dalle precedenti specifiche. **Gestire le eventuali situazioni di errore**.

USCITA CHE DEVE PRODURRE IL PROGRAMMA

```
Test del costruttore e dell'operatore di uscita
POSTAZIONE1:<libera>
POSTAZIONE2:<libera>
POSTAZIONE3:<libera>
POSTAZIONE4:<libera>

Test della aggiungi
POSTAZIONE1:Utente1
POSTAZIONE2:<libera>
POSTAZIONE3:<libera>
POSTAZIONE4:<libera>

POSTAZIONE1:Utente1
POSTAZIONE2:Utente2
POSTAZIONE3:Utente3
POSTAZIONE4:<libera>

Test della elimina (viene liberata la seconda postazione)
POSTAZIONE1:Utente1
POSTAZIONE2:<libera>
POSTAZIONE3:Utente3
POSTAZIONE4:<libera>

Test del costruttore di copia
POSTAZIONE1:Utente1
POSTAZIONE2:<libera>
POSTAZIONE3:Utente3
POSTAZIONE4:<libera>

Test del distruttore (a2 e' stata appena distrutta)

Test dell'operatore di negazione logica
(prima dell'ordinamento)
POSTAZIONE1:Utente1
POSTAZIONE2:Zenone
POSTAZIONE3:Utente3
POSTAZIONE4:Achille
(dopo l'ordinamento)
POSTAZIONE1:Achille
POSTAZIONE2:Utente1
POSTAZIONE3:Utente3
POSTAZIONE4:Zenone
```