PROVA PRATICA 04 LUGLIO 2014

FONDAMENTI DI INFORMATICA I
FOND. DI INFORMATICA E PROGRAMMAZIONE A OGGETTI

Una `TabTreni` è un tabellone in cui vengono visualizzati i treni in partenza di una stazione ferroviaria. Per ciascun treno in partenza, il tabellone deve poter visualizzare l'ora di partenza, il minuto di partenza, e la destinazione (quest'ultima rappresentata da una stringa di lunghezza massima di 30 caratteri).
Il tabellone può contenere un numero limitato di treni.
Utilizzando la rappresentazione interna che **minimizzi il più possibile** il tempo richiesto per **l'inserimento e l'estrazione** di treni dal tabellone, implementare la classe `TabTreni` in modo tale da potervi effettuare le seguenti operazioni:

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

- ✓ **`TabTreni t;`**
  Costruttore che crea un tabellone di treni in partenza di dimensione 5 (ossia tale da poter visualizzare contemporaneamente un massimo di 5 treni). Inizialmente non ci sono treni in partenza.
- ✓ **`TabTreni t(dim);`**
  Costruttore che crea un tabellone di treni di dimensione `dim`. Inizialmente non ci sono treni in partenza.
- ✓ **`t.nuovoTreno(ore, minuti, destinazione);`**
  Operazione che aggiunge al tabellone un treno in partenza alle ore `ore`:`minuti` (`ore` è un intero che va da 0 a 23, `minuti` un intero da 0 a 59).
  Se l'orario di partenza del treno che si vuole inserire è precedente a quella di partenza di un treno presente nel tabellone l'operazione non ha successo: il treno **non** viene inserito e la funzione restituisce `false`. Nei casi di successo restituisce `true`.
- ✓ **`cout<<t;`**
  Operatore di uscita per il tipo `TabTreni`. Un esempio di uscita è il seguente:

  ```
  TRENI IN PARTENZA
  [10:30]TORINO
  [10:45]MILANO
  ```

  dove viene mostra un tabellone con due treni in partenza (non vanno aggiunti spazi bianchi di separazione; i treni sono elencati in ordine di partenza). Qualora non ci fossero treni stamperebbe "TRENI IN PARTENZA" e basta.
- ✓ **`t.partenzaTreno();`**
  Operazione che elimina dal tabellone il treno con partenza più imminente, qualora sia presente almeno un treno (nel qual caso restituisce `true`). Altrimenti lascia il tabellone invariato e restituisce `false`.

SECONDA PARTE ***(si invita a mettere sotto commento le operazioni di questa seconda parte che dovessero impedire la compilazione, il collegamento o la corretta esecuzione del codice)***

- ✓ **`int();`**
  Operatore di conversione di un tabellone in un intero. Restituisce il numero di treni attualmente in partenza presenti nel tabellone.
- ✓ **`TabTreni(const TabTreni &t);`**
  Costruttore di copia per la classe tabellone di treni.

- ✓ **`char*();`**
  Operatore di conversione di un tabellone in un array di caratteri. Restituisce un nuovo array composto dalla concatenazione di tutte le destinazioni per i treni attualmente in partenza, dove ciascuna destinazione è preceduta dai caratteri "=>".
  Esempio. Sia `t` il seguente tabellone:

  ```
  TRENI IN PARTENZA
  [10:45]MILANO
  [11:16]ROMA
  [12:25]VENEZIA
  ```

  L'operazione `cout<<(char*)t;` deve stampare la seguente stringa di caratteri:

  =>MILANO=>ROMA=>VENEZIA

  Nel caso di tabellone vuoto deve restituire la coppia di caratteri => e basta.

- ✓ **`~TabTreni();`**
  Distruttore.

Mediante il Linguaggio C++, realizzare il tipo di dato astratto `TabTreni`, definito dalle precedenti specifiche. **Individuare le eventuali situazioni di errore e metterne in opera un corretto trattamento**.

NOTE SULLO SVOLGIMENTO DELLA PROVA PRATICA

AVVIO E IDENTIFICAZIONE

• Avviare la macchina in modalità diskless, scegliere "Fondamenti di Informatica I" ed effettuare il login: **nome**: studenti **password**: studenti

• Aprire un terminale e al prompt spostarsi sulla cartella 'elaborato' (`$ cd ~/elaborato`). Si utilizzi il comando `pwd` per verificare che ci si trovi nella cartella corretta `/home/studenti/elaborato`.

• Sempre al prompt dare il comando `ident`, sempre da dentro la cartella. Lo script richiede i propri dati (cognome, nome, numero di matricola e password (la password **non va dimenticata** in quanto è indispensabile per scaricare da internet il proprio elaborato a consegna avvenuta). Il comando `ident` crea il file *matricola.txt* nella cartella corrente. Lo script può essere lanciato più volte, in tal caso il file *matricola.txt* viene sovrascritto. Per verificare che il file sia stato creato e che il contenuto sia quello giusto dare il comando (la password è codificata):

```
$ cat /home/studenti/elaborato/matricola.txt
```

• A questo punto il docente verifica che tutti gli studenti abbiamo effettuato l'identificazione, dopodiché provvede a inviare i seguenti file nella cartella `elaborato` del proprio PC: *compito.h, compito.cpp, main.cpp.*
Controllare pertanto che questi file, insieme al file *matricola.txt*, siano presenti sul proprio elaboratore.

SVOLGIMENTO DELLA PROVA

• Definire ed implementare il tipo di dato astratto richiesto e le relative funzioni nei file *compito.h* e *compito.cpp*. Il file *main.cpp* contiene la funzione principale `main()` ed è utilizzato dallo studente per testare la sua implementazione della classe. Il file *main.cpp* può essere modificato a piacere. In sede di valutazione dell'elaborato verrà considerato **esclusivamente il contenuto dei file compito.h e compito.cpp** ed è pertanto **vietato cambiare nome a tali file**.
Per compilare e linkare dare il comando:

`$ g++ main.cpp compito.cpp` (eseguibile invocabile tramite `$ ./a.out`)
(utilizzare `g++ -g` per includere le informazioni di debug qualora si intenda debuggare con `ddd`).

PER CONSEGNARE O RITIRARSI

Recarsi dal docente avendo preso nota dell'identificativo della macchina (g34, s23, …).