Programmazione ad oggetti – Laurea in Informatica
Appello d'Esame – 25/6/2013

Nome..... Cognome.....

Matricola..... Laurea in.....

È VIETATO l'uso di appunti, libri e qualsiasi dispositivo elettronico. Scrivere CHIARAMENTE le soluzioni nel foglio a quadretti.

Quesito 1

Definire una unica gerarchia di classi che includa: (1) una classe base polimorfa `A` alla radice della gerarchia; (2) una classe derivata astratta `B`; (3) una sottoclasse `C` di `B` che sia concreta; (4) una classe `D` definita mediante derivazione multipla con base virtuale.

Quesito 2

Definire un template di classe `Stack<T>` i cui oggetti rappresentano una struttura dati stack (a pila) per elementi di uno stesso tipo `T`. Si ricorda che uno stack implementa la politica LIFO (Last In First Out) di inserimento/estrazione degli elementi: gli elementi vengono estratti in ordine inverso rispetto a quello in cui sono stati inseriti. Il template `Stack<T>` deve soddisfare i seguenti vincoli:

1. `Stack<T>` non può usare i contenitori STL come campi dati (inclusi puntatori e riferimenti a contenitori STL).
2. Il parametro di tipo del template `Stack<T>` ha come valore di default `int`.
3. Gestione della memoria senza condivisione.
4. Deve essere disponibile un costruttore di default che costruisce lo stack vuoto.
5. Deve essere disponibile un costruttore `Stack(int k, const T& t)` che costruisce uno stack contenente `k` copie dell'elemento `t`.
6. Deve essere disponibile un metodo `bool isEmpty()` con il seguente comportamento: `s.isEmpty` ritorna `true` se lo stack `s` è vuoto, `false` altrimenti.
7. Deve essere disponibile un metodo `void push(const T&)` con il seguente comportamento: `s.push(t)` inserisce l'elemento `t` al top dello stack `s`.
8. Deve essere disponibile un metodo `T pop()` con il seguente comportamento: se lo stack `s` non è vuoto, `s.pop()` rimuove l'elemento al top dello stack `s` e lo ritorna; se invece `s` è vuoto allora solleva una eccezione di tipo `Empty` (una classe di eccezioni di cui è richiesta la definizione).
9. Deve essere disponibile un metodo `T* top()` con il seguente comportamento: se lo stack `s` non è vuoto, `s.top()` ritorna un puntatore all'elemento al top dello stack `s`; se invece `s` è vuoto ritorna il puntatore nullo.
10. `Stack<T>` rende disponibile un tipo iteratore `Stack<T>::iterator` i cui oggetti permettono di iterare sugli elementi di uno stack.
11. Opportuno overloading dell'operatore di uguaglianza.
12. Opportuno overloading dell'operatore di indicizzazione.
13. Opportuno overloading dell'operatore di output.

NB: Scrivere la soluzione **chiaramente** nel foglio a quadretti. Per comodità di correzione, definire tutti i metodi inline.

Quesito 3

Si assumano le seguenti specifiche riguardanti la libreria Qt.

- Un oggetto della classe `QString` rappresenta una stringa di caratteri Unicode. La classe `QString` fornisce un costruttore `QString(const char*)` con il seguente comportamento: `QString(str)` costruisce una `QString` inizializzata con la stringa ASCII `str`.
- La classe `QPaintDevice` è la classe base di tutti gli oggetti che possono essere "dipinti" sullo schermo.
  - La classe `QPaintDevice` è polimorfa.
  - La classe `QPaintDevice` rende disponibile un metodo `int height() const` con il seguente comportamento: `pd.height()` ritorna l'altezza in pixel del `QPaintDevice pd`. È inoltre disponibile un metodo `int width() const` con analogo comportamento per la larghezza.
- `QWidget` è una sottoclasse di `QPaintDevice` i cui oggetti rappresentano delle componenti di una interfaccia grafica Qt.
  - La classe `QWidget` rende disponibile un metodo `bool hasFocus() const` con il seguente comportamento: `w.hasFocus()` ritorna `true` quando la componente `w` detiene il keyboard focus.
  - La classe `QWidget` rende disponibile un metodo `void clearFocus()` con il seguente comportamento: `w.clearFocus()` toglie il keyboard focus alla `QWidget w`.
- `QAbstractButton` è derivata direttamente da `QWidget` ed è la classe base astratta dei widget pulsante.
  - La classe `QAbstractButton` rende disponibile un metodo `void setText(const QString&)` con il seguente comportamento: `b.setText(s)` setta l'etichetta testuale del `QAbstractButton b` alla stringa `s`.

Definire una funzione

```
vector<QAbstractButton*> fun(const vector<QPaintDevice*>&)
```

con il seguente comportamento: in ogni invocazione `fun(v)`:

1. per ogni puntatore `p` contenuto nel vector `v`:
   - se `p` punta ad un oggetto che è un `QWidget` con altezza o larghezza $>$ a 50 pixel allora lancia una eccezione di tipo `QString` che rappresenta la stringa "`TooBig`";
   - se `p` punta ad un oggetto `obj` che è un `QWidget` avente sia l'altezza che la larghezza $\leq$ a 50 pixel e che detiene il keyboard focus allora toglie il keyboard focus a `obj`;
   - se `p` punta ad un oggetto `obj` che è un `QAbstractButton` allora setta l'etichetta testuale di `obj` alla stringa "`Pulsante`".
2. l'invocazione `fun(v)` deve ritornare un vector contenente **tutti e soli** i puntatori `p` contenuti nel vector `v` che puntano ad un oggetto che è un `QAbstractButton`.