Programmazione ad oggetti – Laurea in Informatica
Appello d'Esame – 17/3/2014

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

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

È VIETATO l'uso di appunti, libri e qualsiasi dispositivo elettronico. Dove previsto scrivere CHIARAMENTE la risposta nell'apposito spazio.

Quesito 1

Si considerino le seguenti definizioni.

```
class Z {
private:
  int x;
};

class B {
private:
  Z x;
};

class D: public B {
private:
  Z y;
public:
  // ridefinizione di operator=
  ...
};
```

Ridefinire l'assegnazione `operator=` della classe `D` in modo tale che il suo comportamento coincida con quello dell'assegnazione standard di `D`.

SOLUZIONE

Quesito 2

Si consideri il seguente modello di realtà concernente l'azienda Zoox$^{\copyright}$che fornisce un servizio di immagazzinamento di abbigliamento.

1. Definire la seguente gerarchia di classi.
   (a) Definire una classe `Capo` i cui oggetti rappresentano un capo di abbigliamento che Zoox$^{\copyright}$può immagazzinare. Ogni `Capo` è caratterizzato dal designer (rappresentabile mediante una semplice stringa) e dalla taglia. Dotare la classe `Capo` di opportuno/i costruttore/i.
   (b) Definire una classe `TShirt` derivata da `Capo` i cui oggetti rappresentano un capo di abbigliamento di tipologia t-shirt. Ogni `TShirt` è caratterizzata dall'avere la manica lunga o corta. Dotare la classe `TShirt` di opportuno/i costruttore/i.
   (c) Definire una classe `Jeans` derivata da `Capo` i cui oggetti rappresentano un capo di abbigliamento di tipologia jeans. Ogni `Jeans` è caratterizzato dalla larghezza del fondo in cm. Dotare la classe `Jeans` di opportuno/i costruttore/i.
2. Definire una classe `Zoox` i cui oggetti rappresentano un magazzino di Zoox$^{\copyright}$. Più precisamente, un oggetto `Zoox` è caratterizzato dai capi di abbigliamento presenti nel magazzino, che sono rappresentati mediante un opportuno contenitore di puntatori al tipo `Capo`. Naturalmente, il magazzino dovrà poter contenere più articoli di uno stesso capo: ad esempio tre articoli dello stesso jeans di marca Diesel, taglia 48 e con larghezza del fondo 18 cm.

   Devono essere disponibili le seguenti funzionalità:
   (a) un metodo `int giacenza(const Capo&)` con il seguente comportamento: una invocazione `z.giacenza(c)` ritorna il numero di articoli del capo di abbigliamento `c` presenti nel magazzino Zoox$^{\copyright}$`z`.
   (b) un metodo `vector<Jeans> getJeans(string, int, double)` con il seguente comportamento: una invocazione `z.getJeans(marca,size,x)` ritorna un vector (eventualmente vuoto) contenente una copia di tutti i jeans presenti nel magazzino Zoox$^{\copyright}$`z` il cui designer è uguale a `marca`, di taglia `size` e che hanno una larghezza del fondo $\geq$ a `x` cm.
   (c) un metodo `void scarica(const TShirt*)` con il seguente comportamento: una invocazione `z.scarica(pt)` elimina dal magazzino Zoox$^{\copyright}$`z` un articolo della t-shirt `*pt` se `*pt` è presente nel magazzino in almeno un articolo; altrimenti viene sollevata una eccezione `Exc("assente")`. `Exc` è una classe di eccezioni da definire allo scopo.
   (d) un metodo `void insert(const TShirt&, int)` con il seguente comportamento: una invocazione `z.insert(t,num)` aggiunge al magazzino Zoox$^{\copyright}$`z` un numero `num` di articoli della t-shirt `t` se il magazzino non contiene già una t-shirt a manica lunga dello stesso designer di `t`; altrimenti, viene sollevata una eccezione `Exc("presente")`.

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

Quesito 3

Definire un template di classe `dList<T>` i cui oggetti rappresentano una struttura dati **lista doppiamente concatenata** (doubly linked list) per elementi di uno stesso tipo `T`. Il template `dList<T>` deve soddisfare i seguenti vincoli:

1. Gestione della memoria senza condivisione.
2. `dList<T>` rende disponibile un costruttore `dList(int k, const T& t)` che costruisce una lista contenente `k` nodi ed ognuno di questi nodi memorizza una copia di `t`.
3. `dList<T>` permette l'inserimento in testa ed in coda ad una lista in tempo $O(1)$ (cioè costante):
   - Deve essere disponibile un metodo `void insertFront(const T&)` con il seguente comportamento: `dl.insertFront(t)` inserisce l'elemento `t` in testa a `dl` in tempo $O(1)$.
   - Deve essere disponibile un metodo `void insertBack(const T&)` con il seguente comportamento: `dl.insertBack(t)` inserisce l'elemento `t` in coda a `dl` in tempo $O(1)$.
4. `dList<T>` rende disponibile un opportuno overloading di `operator<` che implementa l'ordinamento lessicografico (ad esempio, si ricorda che per l'ordinamento lessicografico tra stringhe abbiamo che `"campana" < "cavolo"` e che `"eccellente" < "ottimo"`).
5. `dList<T>` rende disponibile un tipo iteratore costante `dList<T>::const_iterator` i cui oggetti permettono di iterare sugli elementi di una lista.

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