Laurea in Informatica – Programmazione ad Oggetti – Appello d'Esame 17/2/2016

Nome..... Cognome..... Matricola.....

È VIETATO l'uso di oggetti diversi dalla penna. Scrivere le soluzioni CHIARAMENTE nel foglio a quadretti.

Esercizio 1

Si **assumano** le seguenti specifiche riguardanti la libreria Qt (**attenzione:** non si tratta di codice da definire!).

- `QWidget` è la classe base di tutte le classi Gui della libreria Qt.
  - La classe `QWidget` ha il distruttore virtuale.
  - La classe `QWidget` rende disponibile un metodo virtuale `QSize sizeHint() const` con il seguente comportamento: `w.sizeHint()` ritorna un oggetto di tipo `QSize` che rappresenta la dimensione raccomandata per il widget `w`. È disponibile l'operatore esterno di uguaglianza `bool operator==(const QSize&, const QSize&)` che testa l'uguaglianza tra oggetti di `QSize`.
  - La classe `QWidget` rende disponibile un metodo virtuale di clonazione `QWidget* clone()` con l'usuale contratto di "costruttore di copia polimorfo": `pw->clone()` ritorna un puntatore polimorfo ad nuovo oggetto `QWidget` che è una copia polimorfa di `*pw`. Ogni sottoclasse di `QWidget` definisce quindi il proprio overriding di `clone()`.
- La classe `QAbstractButton` deriva direttamente e pubblicamente da `QWidget` ed è la classe base astratta di tutti i button widgets.
  - Le classi `QCheckBox`, `QPushButton` e `QRadioButton` derivano direttamente e pubblicamente da `QAbstractButton`. Le classi `QCheckBox` e `QPushButton` definiscono il proprio overriding di `QWidget::sizeHint()`.
- La classe `QAbstractSlider` deriva direttamente e pubblicamente da `QWidget` ed è la classe base astratta di tutti gli slider widgets.
  - Le classi `QScrollBar` e `QSlider` derivano direttamente e pubblicamente da `QAbstractSlider`. Entrambe le classi definiscono il proprio overriding di `QWidget::sizeHint()`.

Definire una funzione `vector<QAbstractButton*> fun(list<QWidget*>&, const QSize&)` con il seguente comportamento: in ogni invocazione `fun(lst,sz)`

1. per ogni puntatore `p` elemento (di tipo `QWidget*`) della lista `lst` al momento del passaggio del parametro:
   (a) se `*p` ha una dimensione raccomandata uguale a `sz` allora inserisce nella lista `lst` un puntatore ad una copia di `*p`;
   (b) se `p` non è nullo, `*p` non è uno slider widget e ha una dimensione raccomandata uguale a `sz` allora rimuove dalla lista `lst` il puntatore `p` e dealloca l'oggetto `*p`;
2. inoltre, si devono rimuovere dalla lista `lst` al momento del passaggio del parametro ed inserirli nel vector di `QAbstractButton*` da ritornare tutti i rimanenti puntatori `p` della lista `lst` tali che `*p` è un `QCheckBox` oppure un `QPushButton`.

Esercizio 2

Definire un template di classe `Array<T>` i cui oggetti rappresentano una struttura dati "array ridimensionabile" di elementi di uno stesso tipo `T`. Si ricorda che in un array ridimensionabile la sua dimensione (cioè il numero di elementi correntemente contenuti) è sempre $\leq$ alla sua capacità (cioè il numero di elementi che può contenere senza dover ridimensionarsi), e quando si inserisce un nuovo elemento in un array con dimensione uguale alla capacità, l'array viene ridimensionato raddoppiandone la capacità. Il template `Array<T>` deve soddisfare i seguenti vincoli:

1. Ovviamente, `Array<T>` non può usare i contenitori STL o Qt come campi dati (inclusi puntatori e riferimenti a contenitori STL o Qt).
2. Deve essere disponibile un costruttore `Array(int k = 0 , const T& t = T())` che costruisce un array contenente `k` copie di `t` quando $k > 0$, mentre se $k \leq 0$ costruisce un array vuoto.
3. Gestione della memoria senza condivisione.
4. Deve essere disponibile un metodo `void pushBack(const T&)` con il seguente comportamento: `a.pushBack(t)` inserisce l'elemento `t` alla fine dell'array `a` dopo il suo ultimo elemento corrente; ciò provoca quindi il ridimensionamento di `a` se e solo se la dimensione di `a` è uguale alla sua capacità.
5. Deve essere disponibile un metodo `T popBack()` con il seguente comportamento: se l'array `a` non è vuoto, `a.popBack()` rimuove l'ultimo elemento corrente di `a` e quindi lo ritorna; se invece `a` è vuoto allora solleva una eccezione di tipo `Empty` (una opportuna classe di eccezioni di cui è richiesta la definizione).
6. Opportuno overloading dell'operatore di uguaglianza.
7. Opportuno overloading dell'operatore di output.

Esercizio 3

```
class Z {
 public: Z(int x) {}
};

class A {
 public:
  void f(int){cout << "A::f(int) ";}
  virtual void f(bool){cout <<"A::f(bool) ";}
  virtual void f(Z){cout <<"A::f(Z) ";}
};

class B: virtual public A {
 public:
  void f(const bool&){cout<< "B::f(const bool&) ";}
  void f(const int&){cout<< "B::f(const int&) ";}
};

class C: virtual public A {
 public:
  virtual void f(Z){cout <<"C::f(Z) ";}
};

class D: public B, public C {
 public:
  virtual void f(int*){cout<< "D::f(int*) ";}
  void f(int&){cout <<"D::f(int&) ";}
};

class E: public D {
 public:
  void f(Z){cout <<"E::f(Z) ";}
};

B* pb=new B; C* pc = new C; D* pd = new D; E* pe = new E; A *pa1=pb, *pa2=pc, *pa3=pd, *pa4=pe; C *pc1=pe;
```

Le precedenti definizioni compilano correttamente. Per ognuna delle seguenti istruzioni scrivere nell'apposito spazio:

- **NON COMPILA** se la compilazione dell'istruzione provoca un errore;
- **ERRORE RUN-TIME** se l'istruzione compila correttamente ma la sua esecuzione provoca un errore a run-time;
- se l'istruzione compila correttamente e non provoca errori a run-time allora si scriva la stampa che l'esecuzione produce in output su `std::cout`; se non provoca alcuna stampa allora si scriva **NESSUNA STAMPA**.

```
(dynamic_cast<B*>(pa1))->f(1);      .....

(dynamic_cast<B*>(pa1))->f(true);   .....

pa1->f(true);                       .....

pa2->f(1);                          .....

(dynamic_cast<C*>(pa2))->f(1);      .....

(dynamic_cast<E*>(pa2))->f(1);      .....

(dynamic_cast<C*>(pa3))->f(0);      .....

(dynamic_cast<D*>(pa3))->f(0);      .....

pa4->f(1);                          .....

(dynamic_cast<C*>(pa4))->f(1);      .....

pc1->f(1);                          .....

(static_cast<E*>(pc1))->f(1);       .....

(static_cast<A*>(pc1))->f(1);       .....
```