UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea in Informatica
Prova di Programmazione ad Oggetti
Data: 19 luglio 2006
Tempo: 2 ore e 30 minuti

Esercizio 1

Siano date le seguenti dichiarazioni di classe:

```
class Volo {
      public:
            Volo();
            ➔ Volo(const MyString& p, const MyString& a,
                  int cod, const MyString& comp, double costo);
            ➔ bool operator==(const Volo& volo) const;
            bool isEmpty() const;
            MyString getPartenza() const;
            MyString getArrivo() const;
            int getCodiceAereo() const;
            MyString getCompagnia() const;
            double getCosto() const;
      private:
            MyString partenza;
            MyString arrivo;
            int codiceAereo;
            MyString compagnia;
            double costo;
};

class ListaVoli {
      public:
            ListaVoli(int size = 0);
            ➔ ListaVoli(const ListaVoli& lista_voli);
            ➔ ~ListaVoli();
            const ListaVoli& operator=(const ListaVoli& lista_voli);
            bool insert(const Volo & volo);
            bool remove(const Volo & volo);
            int ePresente(const Volo & volo) const;
            ListaVoli cerca(int codiceAereo) const;
      private:
            Volo *voli;
            int size;
};
```

Implementare le funzioni delle classi **`Volo`** e **`ListaVoli`** indicate dalla freccia. Indicare **ove necessario** eventuali istruzioni di **`assert`**.

NOTE: Un **`Volo`** è caratterizzato da una località di partenza ed una di destinazione, dal codice dell'aereo, dal nome della compagnia di volo e dal costo del volo. Una **`ListaVoli`** è un archivio contenente un certo numero di voli.

Esercizio 2

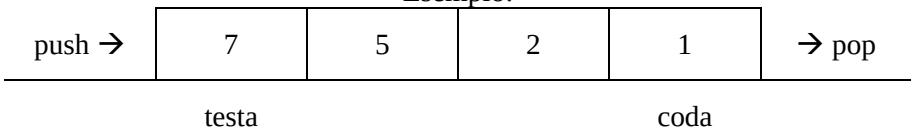
Sia data le seguente dichiarazione di classe:

```
class CodaInt : protected ListaInt
{
      public:
            void push(int elem); //inserimento in testa
            void pop(); //estrazione dalla coda
            int top(); //restituisce l'elemento in coda
            bool isEmpty() const;
};
```

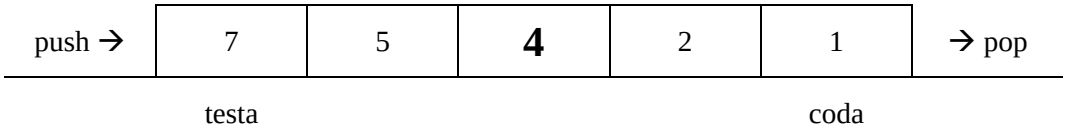
- Progettare le interfacce di una classe **`CodaPrioritaria`** usando l'ereditarietà a partire dalla classe **`CodaInt`**.
- Implementare tutti i metodi necessari per la classe **`CodaPrioritaria`** affinché sia completamente funzionante (considerare già implementati tutti i metodi di **`ListaInt`** e **`CodaInt`**)

NOTE: Una coda di interi è una specializzazione dei una lista di interi dove l'ultimo ad entrare nella lista sarà l'ultimo ad uscire, ovvero si inserisce in testa (push) e si estrae dalla coda (pop). Una coda prioritaria di interi è una particolare coda dove all'atto dell'inserimento in testa (push) se il valore numerico da inserire è maggiore dell'ultimo elemento (quello attualmente in testa) il comportamento è identico a quello della coda. Se invece l'elemento da inserire è più piccolo verrà inserito subito prima del primo elemento che rispetto ad esso è maggiore o uguale.

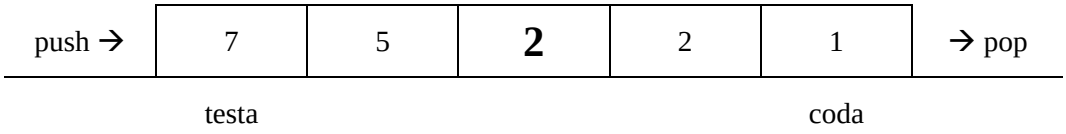
Esempio:



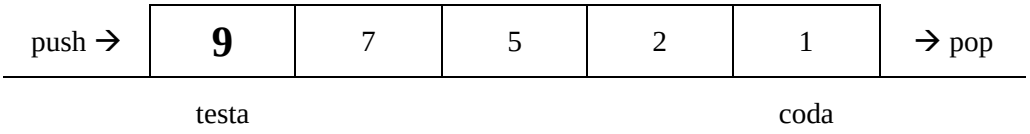
se volessimo inserire l'elemento **4** avremmo:



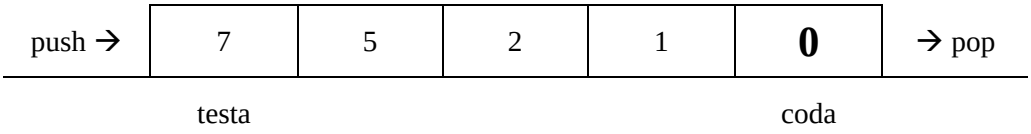
se volessimo inserire l'elemento **2** avremmo:



se volessimo inserire l'elemento **9** avremmo:



se volessimo inserire l'elemento **0** avremmo:



Soluzione

```
Volo::Volo(const MyString& p, const MyString& a, int cod, const MyString& comp, double costo)
: partenza(p), arrivo(a), codiceAereo(cod), compagnia(comp), costo(costo)
{
     //Le assert servono solo ad assicurare che i valori siano consistenti
     assert(p.getLength() > 0);
     assert(a.getLength() > 0);
     assert(cod > 0);
     assert(comp.getLength() > 0);
     assert(costo > 0);
}
//oppure
Volo::Volo(const MyString& p, const MyString& a, int cod, const MyString& comp, double costo)
{
     //Le assert servono solo ad assicurare che i valori siano consistenti
     assert(p.getLength() > 0);
     assert(a.getLength() > 0);
     assert(cod > 0);
     assert(comp.getLength() > 0);
     assert(costo > 0);

     partenza = p;
     arrivo = a;
     codiceAereo = cod;
     compagnia = comp;
     this->costo = costo;
}


//-----------------------------------

Vool Volo::operator==(const Volo& volo) const
{
     return (volo.partenza == partenza &&
               volo.arrivo == arrivo &&
               volo.codiceAereo == codiceAereo &&
               volo.compagnia == compagnia &&
               volo.costo == costo);
}

//-----------------------------------

ListaVoli::ListaVoli(const ListaVoli& lista_voli) : size(lista_voli.size)
{
     voli = new Volo[size];
     for(int i=0; i<size; i++)
          voli[i] = lista_voli.voli[i];
}

//-----------------------------------

ListaVoli::~ListaVoli()
{
     delete []voli;
}
```

```
#ifndef CODA_PRIORITARIA_H
#define CODA_PRIORITARIA_H

#include "Coda.h"

class CodaPrioritaria: public CodaInt
{
public:
     void push(int);
};

#endif
```

```
//----------------------------------

#include "CodaPrioritaria.h"

void CodaPrioritaria::push(int elem)
{
     NodoInt *prec = NULL;
     NodoInt *curr = testa;

     while(curr != NULL)
     {
          if(curr->getValore() <= elem)
               break;
          prec = curr;
          curr = curr->nextNodo();
     }
     if(prec == NULL)
          addInTesta(elem);
     else
          addAfter(elem, prec);
}
//oppure
void CodaPrioritaria::push(int elem)
{
     NodoInt *curr = testa;

     if(curr == NULL || curr->getValore() <= elem)
          addInTesta(elem);
     else
     {    while(curr->nextNodo() != NULL)
          {
               if(curr->nextNodo()->getValore() <= elem)
                    break;
               curr = curr->nextNodo();
          }
          addAfter(elem, curr);
     }
}
```