UNIVERSITÀ DELLA CALABRIA

Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea in Informatica
Prova di Programmazione ad Oggetti
Data: 20 settembre 2006
Tempo: 2 ore

Esercizio 1

Si vuole implementare un sistema per la gestione delle partenze e degli arrivi in una **`Stazione`** ferroviaria.

- Per ogni **`Treno`** si vuole memorizzare il numero dei posti, un codice e la sua tipologia.
- Una **`Stazione`** tiene traccia delle informazioni:
  - sui treni che in essa partono/arrivano (utilizzando un vettore **`treni`** e relativa dimensione)
  - degli orari delle partenze dei treni (attraverso un vettore **`partenze`**)
  - degli orari degli arrivi dei treni (attraverso un vettore **`arrivi`**)
- Gli orari sono memorizzati facendo uso della classe **`Orario`**

Le seguenti classi consentono di implementare i requisiti su elencati.

```
class Orario {
   public:
      Orario();
      Orario(int ore, int minuti);
      bool operator<=(const Orario & o) const;
      bool operator==(const Orario & o) const;
      bool operator>=(const Orario & o) const;
      int getOre() const;
      int getMinuti() const;
   private:
      int sec;
};

enum TipoTreno {INDEF, REG, INTERREG, INTERCITY, EUROSTAR};

class Treno {
   public:
      Treno();
      ➔ Treno(int np, const MyString & cod, TipoTreno tipo);
      int getPosti() const;
      const MyString & getCodice() const;
      TipoTreno getTipo() const;
      void setPosti (int np);
      void setCodice (const MyString & cod);
      void setTipo (TipoTreno tipo);
   private:
      int numPosti;  MyString codice;  TipoTreno tipo;
};

class Stazione {
   public:
      Stazione();
      ➔ Stazione(Treno t[], int numTreni);
      ➔ ~ Stazione();
      setPartenza (const MyString & codTreno, const Orario & or);
      setArrivo (const MyString & codTreno, const Orario & or);
      Orario getPartenza (const MyString & codTreno) const;
      Orario getArrivo (const MyString & codTreno) const;
      ➔ bool esisteTrenoInPartenza (TipoTreno t, const Orario & o1,
                                    const Orario & o2) const;
   private:
      Treno *treni;  int numeroTreni;  Orario *partenze;  Orario *arrivi;
};
```

Si supponga che inizialmente gli orari di partenza/arrivo siano sconosciuti (per fissarli è necessario usare i relativi metodi **`set`**). Si assuma inoltre che un certo treno passi (arrivi e riparta) *una sola volta al giorno* dalla stazione e, quindi, abbia un *solo orario di arrivo ed un solo orario di partenza*. Il metodo **`esisteTrenoInPartenza`** dice se esiste *almeno un treno* che parte dalla stazione in un certo intervallo di tempo.

Implementare i 4 metodi delle classi **`Treno`** e **`Stazione`** indicati dalle frecce. Si considerino implementati correttamente tutti gli altri metodi.

Esercizio 2

Un insieme è una **collezione** di elementi **senza duplicati**. Una struttura dati per rappresentare un **insieme dinamico** (possibilità di inserire e/o eliminare elementi dinamicamente) può essere implementata mediante una **lista concatenata**. Per rendere **efficienti** le operazioni sull'insieme è possibile mantenerne **ordinati** gli elementi al suo interno.
Si **progetti** la classe **`InsiemeOrdinato`** ad elementi interi usando l'ereditarietà a partire dalla classe **`ListaInt`**. Si presti attenzione al tipo di derivazione (pubblica, protetta o privata) tenendo presente che la nuova classe dovrà **esclusivamente** avere:

- Un costruttore di default
- Un costruttore di copia
- Un metodo di inserimento per singolo elemento
- Un metodo che unisce duen insiemi (l'insieme corrente con un secondo insieme)
- Un metodo di eliminazione per singolo elemento
- Un metodo che verifichi se un elemento è presente nell'insieme
- Un metodo che restituisca l'elemento più piccolo dell'insieme
- Un metodo che restituisca l'elemento più grande dell'insieme
- Un metodo che restituisca il numero di elementi dell'insieme
- Un metodo che ci dice se l'insieme è vuoto
- Un operatore di stampa su standard output

Si implementino l'interfaccia della classe e **solo** i 2 metodi sottolineati. Si considerino implementati correttamente tutti gli altri metodi.

NOTE: Nell'implementare il metodo che implementa l'unione di due insiemi si tenga presente che:

- L'insieme risultante **non** dovrà contenere **duplicati**
- L'insieme risultante dovrà essere **ordinato**
- Il fatto che i due insiemi di partenza siano ordinati può essere sfruttato per implementare efficientemente l'operazione
- Si **consiglia** l'uso degli iteratori

SOLUZIONE 1

```
Treno::Treno(int np, const MyString & cod, TipoTreno tipo) :
   numPosti(np), codice(cod), tipo(tipo) {}


Stazione::Stazione(Treno t[], int numTreni)
{
   numeroTreni = numTreni;
   treni = new Treno[numTreni];
   for(int i = 0; i < numTreni; i++)
      treni[i] = t[i];
   partenze = new Orario[numTreni];
   arrivi = new Orario[numTreni];
}


Stazione::~Stazione()
{
   delete []treni;
   delete []partenze;
   delete []arrivi;
}

bool Stazione::esisteTrenoInPartenza (TipoTreno t, const Orario & o1,
                                      const Orario & o2) const
{
   for (int i = 0; i < numeroTreni; i++)
      if(treni[i].getTipo() == t)
         if(o1 <= partenze[i] && partenze[i] <= o2)
            return true;
   return false;
}
```

SOLUZIONE 2

```
class InsiemeOrdinato : protected ListaInt
{
   friend ostream& operator<<(ostream&, const InsiemeOrdinato&);
       public:
         InsiemeOrdinato();
         InsiemeOrdinato(const InsiemeOrdinato & insieme);
         void inserisci(int val);
         void unisci (const InsiemeOrdinato & insieme);
         void elimina(int val);
         bool esiste (int val) const;
         int getMinimo() const;
         int getMassimo() const;
         int getSize() const;
         bool eVuoto() const;
};



void InsiemeOrdinato::unisci (const InsiemeOrdinato & insieme)
{
   ListaInt *ins1 = this;
   ListaInt *ins2 = (ListaInt*)&insieme;
   InsiemeOrdinato *ins3 = new InsiemeOrdinato();

   Iteratore i1(*ins1), i2(*ins2);
   i1.VaiInTesta(); i2.VaiInTesta();

   while(!i1.isNull() && !i2.isNull())
      if(i1.getCurrentValue() <= i2.getCurrentValue())
         insAndMove(i1,ins3);
      else
         insAndMove(i2,ins3);
   while(!i1.isNull())
      insAndMove(i1,ins3);
   while(!i2.isNull())
      insAndMove(i2,ins3);

   svuota();
   *this = *ins3;
}
void insAndMove(Iteratore & iter, InsiemeOrdinato * io)
{
   int val = iter.getCurrentValue();
   iter.MoveNext();
   if(io->getSize() == 0 || io->valoreInCoda() != val)
      io->addInCoda(val);
}



void InsiemeOrdinato::elimina(int val)
{
   NodoInt* temp = cerca(val);
   if(temp != NULL)
      del(temp);
}
```