```
/****
 *
 * Class Rolodex is the model class for the basic rolodex object, including the
 * edit operations to add, delete, change, and find rolodex entries.
 *
 */

#ifndef roldexIncluded
#define roldexIncluded

#include "add-input-errors.h"
#include "basic-types.h"
#include "card-list.h"
#include "card.h"
#include "change-input-errors.h"
#include "delete-input-error.h"


class Rolodex : public Model {

  public:

    Rolodex(View* v = NULL);
    /*
     * Construct an initially empty rolodex.  Initialize the error objects for
     * each function that throws an exception.
     *
     * post: (exists (CardList* cl) (this’->cl == cl) and (cl->Len() == 0))
     *           and
     *       (exists (AddInputErrors* aie) (this’->aie == aie));
     *           and
     *       (exists (DeleteInputError* die) (this’->die == die));
     *           and
     *       (exists (ChangeInputErrors* cie) (this’->cie == cie));
     */

    ~Rolodex();
    /*
     * Deep destruct this.  I.e., delete all cards in cl.
     *
     * post: freed(this) and freed(n) and freed(addr);
     */

    void Add(Card* c);
    /*
     * Add the given card to this if a card of the same id is not already there
     * and there is room for another card.
     *
     * pre:
     *      // There is no card in the input this with the same id as the given
     *      // input card.
     *      not CardAlreadyThere(c)
     *
     *          and
     *
     *      // The values of the given card fields are valid.
     *      CardIsValid(c)
     *
     *          and
     *
     *      // There is room for another card.
     *      (cl->Len() < MAXCARDS)
     *
     * post:
     *      // If none of the preconditions is violated, then a normal
     *      // postcondition holds, else an exception is thrown.
     *      if (not AddPrecondsViolated(c)) (
     *          // A card is in the output this iff it is in the input this or
     *          // it’s the new card to be added.
     *          forall (Card* c’) this’->In(c’) iff (In(c’) or (c’ == c))
     *
     *              and
     *
     *          // The size of this is increased by 1.
     *          (cl’->Length() == cl->Length() + 1)
     *      )
     *      else (
     *          // An AddInputException is thrown if any of the preconditions
     *          // is violated.  The value of the exception flag is set for
     *          // each violated precondition via the ValidataAddInput
     *          // function.
     *          (throw == aie) and (aie == ValidateAddInput(c));
     */

    void Delete(Id id);
    /*
     * Delete the card of the given id from this.
     *
     * pre:
     *      // There is a card of the given id in the input this.
     *      exists (Card* c’ | In(c’)) c’->GetId() == id;
     *
     * post:
     *      // If the precondition is not violated, then a normal postcondition
     *      // holds, else an exception is thrown.
     *      if (DeletePrecondViolated(c)) then (
     *          // A card is in the output this iff it is in the input this and
     *          // its id is different than the given id.
     *          forall (Card* c’)
     *              this’->In(c’) iff (In(c’) and (c’.id != id));
     *      )
     *      else
     *          // A DeleteInputException is thrown if the precondition is
     *          // violated.
     *          throw == die;
     */

    void Change(Id id, Card* c);
    /*
     * Change the card of the given id in this to the given new card.
     *
     * pre:
```

```
     *      // There is a card of the given id in the input this and that card
     *      // is not identical with the given input card, and the given card
     *      // is valid.
     *      exists (Card* c’ | In(c’) (c’->GetId() == id) and
     *          (c’ != c) and CardIsValid(c);
     *
     * post:
     *      // If the precondition is not violated, then a normal postcondition
     *      // holds, else an exception is thrown.
     *      if (ChangePrecondViolated(c)) then (
     *          // A card is in the output this iff it is in the input this and
     *          // is not the card with the given id in the input this, or it’s
     *          // the given input card.
     *          forall (Card* c’)
     *              this’->In(c’) iff (In(c’) and
     *                  (not exists (Card* c’’ | In(c’’))
     *                      (c’’->GetId() == id) and (c’’ == c’)) or
     *                  (c’ == c))
     *      )
     *      else
     *          // A ChangeInputException is thrown if the precondition is
     *          // violated.
     *          throw == cie;
     *
     */

    CardList* Find(Name* n);
    /*
     * Find all of the cards of the given name in the given rolodex.
     *
     * post:
     *      // Cards in the output list consist of those in the input this with
     *      // the given name.
     *      forall (Card* c)
     *           (cl->In(c) iff (In(c) and (c->GetName() == n))
     *
     *          and
     *
     *      // The output card list is sorted by ascending order of card id.
     *      forall (int i, int j |
     *          (i >= 1) and (i <= cl->Length()) and
     *                  (j >= 1) and (j <= cl->Length()))
     *              if i<j then
     *                  cl->GetNth(i)->GetId() < cl->GetNth(j)->GetId();
     */

    void Print();
    /*
     * Print this to stdout.  In the current design, the specs are the same as
     * Dump below, q.v.  In general, if there is a distinction between Print
     * and Dump it is that the former produces a more nicely formated output
     * than the latter.
     */

    void Dump();
    /*
     * Dump this to stdout by dumping each card, in List::Enum order.
     */



    /**
     * View Service functions.  These are functions that provide data
     * manipulation services to the views that the views cannot know how to
     * implement.
     */

    Sex StringToSex(String* s);
    /*
     * Convert to the given string to one of the Sex enumeration literals.  The
     * String value must be all lower case and spelled correctly, i.e., either
     * "male" or "female".  If s is not one of these values, Female is
     * returned.
     *
     * post: if (s == "male") then return == Male else return == Female
     */


  protected:
    /**
     * Functions.
     */
    AddInputErrors* ValidateAddInput(Card* c);
    /*
     * Validate the given card per the precondition of Add.  Return the
     * appropriately set AddInputErrors value.  See the definition of
     * AddInputErrors for further information.
     */

    Rolodex(Rolodex &);             // Standard protected copy constructor
    Rolodex& operator=(Rolodex&);   // Standard protected op =


    /**
     * Data members.
     */
    CardList* cl;               // Simple list represenation of the card list
    AddInputErrors* aie;        // Error messages for the Add function
    DeleteInputError* die;      // Error message for the Delerte function
    ChangeInputErrors* cie;     // Error messages for the Change function


    /**
     * Auxiliary spec functions.
     *
     * bool CardAlreadyThere(Card* c) {
     *      not (exists (Card* c’ | In(c’))
     *              c’->GetId() == c->GetId())
     * }
     *
     * bool CardIsValid(Card* c) {
     *      // The length of the name is <= 30 characters
     *      (c->GetName()->Length() <= 30);
```

```
     *
     *          and
     *
     *      // The length (i.e, numer of digits) of the id is 9
     *      Stringify(c->GetId())->Length() == 9)
     *
     *          and
     *
     *      // Age is a reasonable range
     *      ((c->GetAge() >= 0) and (c->GetAge() <= 200))
     *
     *          and
     *
     *      // Sex is valid
     *      (c->GetSex() != InvalidSex)
     *
     *          and
     *
     *      // The length of the address is <= 40 chars
     *      (c->GetAddr() < 40)
     *
     * }
     *
     * bool AddPrecondsViolated(Card* c) {
     *     CardAlreadyThere(c) or (not CardIsValid(c));
     * }
     *
     * bool DeletePrecondViolated(Card* c) {
     *     not (exists (Card* c' | In(c')) c'->GetId() == id)
     * }
     *
     * bool ChangePrecondViolated(Card* c) {
     *     not (exists (Card* c' | In(c')) (c'->GetId() == id) and
     *                                     (c' != c));
     * }
     *
     * bool In(Card* c);
     * // Return true if the given card is anywhere in this.
     *     Find(c) != null
     *
     */

};

#endif
```