```
(****
 *
 * This file is a collection of all of the Rolodex objects and operations
 * discussed in the formal specification primer.  Some of the advanced
 * operations are not fully defined, as discussed in the comments.  See the
 * primer for a complete discussion.
 *
 *)



(**
 * Object Rolodex and its subsidiary objects define the basic Rolodex
 * structure.
 *)

object Rolodex is name:Name and cards:Card* and state:SystemState;
object Card is name:Name and id:Id and age:Age and sex:Sex and addr:Address;
object Name is string;
object Id is integer;
object Age is integer;
object Sex is Male or Female;
object Male;
object Female;
object Address is string;


(**
 * Operations Add, Delete, Change, and Find are the basic rolodex operations.
 *)
operation Add(r:Rolodex, c:Card, fs:FileSpace)->(r':Rolodex, fs':FileSpace)

    precondition:

        (* The length of the name is <= 30 characters *)
        (#(c.name) <= 30)

            and

        (* The length (i.e, numer of digits) of the id is 9 *)
        (#(c.id) = 9)

            and

        (* Age is a reasonable range *)
        ((c.age >= 0) and (c.age <= 200))

            and

        (* The length of the address is <= 40 chars *)
        (#(c.addr) < 40)

            and

        (* There is no card in the input Rolodex with the same Id
         * as the given input card. *)
        not (exists (c' in r.cards) c'.id = c.id);


    postcondition:

        (* A card is in the output rolodex iff it is in the input rolodex or
         * it's the new card to be added. *)
        forall (c':Card)
            (c' in r'.cards) iff ((c' in r.cards) or (c' = c))

            and

        (* The previous state of the output rolodex is the card list of the
         * input rolodex.  See the Undo operation for use of the previous
         * state. *)
        (r'.state.prev = r.cards)

            and

      (* The checkpointing state is properly maintained. *)
      if r.state.options.chkpton then
          if r.state.chkpt = 0 then    (* Time to do checkpoint save *)
              (r'.state.chkpt =
                  r.state.options.chkptinterval) and
              (fs' = Save(fs,r))
          else                         (* Not time yet, decrement counter *)
              (r'.state.chkpt = r.state.chkpt - 1) and
              (fs' = fs)
      else
          true;

end Add;


op Delete(r:Rolodex, n:Name)->(r':Rolodex)
    pre:
        (* There is a card of the given name in the input Rolodex *)
        exists (c' in r.cards) c'.name = n;
    post:
        (* Cards in the output Rolodex consist of those
         * in the input Rolodex, except for those with
         * the same name as the given input name. *)
        forall (c':Card)
            (c' in r'.cards) iff ((c' in r.cards) and (c'.name != n));
end;


op Change(r:Rolodex, n:Name, c:Card)->(r':Rolodex)
    pre:
        (* There is a card of the given name in the input Rolodex *)
        exists (c' in r.cards) c'.name = n;
    post:
        (* The given card is in the output Rolodex *)
        (c in r'.cards)

            and
```

```

        (* No other cards of the same name are in the Rolodex *)
        forall (c':Card | c' != c)
            (c' in r'.cards) iff ((c' in r.cards) and (c'.name != n));
end;

op Find(r:Rolodex, n:Name)->(cl:Card*)
    post:
        (* Cards in the output list consist of those
         * in the input Rolodex with the given name *)
        forall (c:Card)
            (c in cl) iff ((c in r.cards) and (c.name = n))

            and

        (* The output card list is sorted by ascending
         * order of card id. *)
        forall (i,j:integer |
           (i in [1..#cl]) and (j in [1..#cl]))
                if i<j then cl[i].id < cl[j].id;
end;



(**
 * Operation Undo undoes the last Add.  It could be extended to undo other
 * operations.
 *)
op Undo(r: Rolodex)->(r':Rolodex)
    post:
        (* If the input Rolodex has a previous card list,
         * then the cards of the output Rolodex are that list
         * and the previous of the output is nil (so only one
         * level of undo is possible).
         * Otherwise, the output Rolodex is the same as the
         * input Rolodex. *)
        if (r.state.prev != nil)
        then (r'.cards = r.state.prev) and (r'.state.prev = nil)
        else r' = r;
end;


(**
 * Object SearchInfo and its subsidiary objects define search patterns used in
 * the extended version of the Find operation defined below.
 *)

object SearchInfo is np:NamePattern and idp:IdPattern and
        ap:AgePattern and sp:SexPattern and adp:AddressPattern
    description: (*
        Each component of SearchInfo is a search pattern that corresponds to
        one of the fields of a card..
    *);
end SearchInfo;

object AgePattern is lessp:AgeLessThan or gtrp:AgeGreaterThan or
        rangep:AgeRange or eqp:Age
    description: (*
        An AgePattern allows the user to search for cards with an age value
        less than a given age, greater than a given age, between a range of
        given ages, equal to a specific age, or with a specific age.
    *);
end AgePattern;

obj NamePattern;     (* Should be defined comparably to AgePattern *)
obj IdPattern;       (* Ibid. *)
obj SexPattern;      (* Ibid. *)
obj AddressPattern;  (* Ibid. *)

object AgeLessThan is lts:LessThanSymbol and age:Age
    description: (*
        This pattern specifies all ages less than a particular age.
    *);
end AgeLessThan;

object AgeGreaterThan is lts:GreaterThanSymbol and age:Age
    description: (*
        This pattern specifies all ages greater than a particular age.
    *);
end AgeGreaterThan;

object AgeRange is age1:Age and rs:RangeSymbol and age2:Age
    description: (*
        This pattern specifies all ages in a range between age1 and age2.
    *);
end AgeRange;

object LessThanSymbol;
object GreaterThanSymbol;
object RangeSymbol;


(**
 * The following version of operation Find takes SearchInfo as input.  Compare
 * it to the earlier definition of Find, that takes just a Name as input.
 *)
operation Find(r:Rolodex, si:SearchInfo)->(cl:Card*)
    post:
        (*
         * All cards in the output list must be found according to the given
         * search info, and the output list must be sorted by Card id.
         *)
        CardsFound(r,si,cl)
            and
        SortedById(cl);
    description: (*
        Find zero or more cards that match the constraints specified in the
        given SearchInfo.
    *);
end Find;

function CardsFound(r:Rolodex, si:SearchInfo, cl:Card*)->boolean =
```

```
    (* Cards in the given card list consist of those, and only those, in the
     * given Rolodex that match the given search info. *)
    forall (c:Card)
        (c in cl) iff ((c in r.cards) and Match(c,si));

function Match(c:Card, si:SearchInfo)->boolean =
        MatchName(c.name, si.np) and
        MatchId(c.id, si.idp) and
        MatchAge(c.age, si.ap) and
        MatchSex(c.sex, si.sp) and
        MatchAddress(c.addr, si.adp);

function MatchAge(age:Age, ap:AgePattern)->boolean =
    if (ap?lessp) then
        age < ap.lessp.age
    else if (ap?gtrp) then
        age > ap.gtrp.age
    else if (ap?rangep) then
        (age >= ap.rangep.age1) and (age <= ap.rangep.age2)
    else if (ap?eqp) then
        age = ap.eqp
    else if (ap = nil) then
        true;

function MatchName(Name, NamePattern)->boolean;           (* To be defined *)
function MatchId(Id, IdPattern)->boolean;                 (* Ibid. *)
function MatchSex(Sex, SexPattern)->boolean;              (* Ibid. *)
function MatchAddress(Address, AddressPattern)->boolean;  (* Ibid. *)

function SortedById(cl:Card*)->boolean =
    forall (i,j:integer |
       (i in [1..#cl]) and (j in [1..#cl]))
            if i<j then cl[i].id < cl[j].id;


(**
 * Object FileSpace and its subsidiary objects define an abstract file space
 * used in the file management operations below.
 *)

obj FileSpace is File*;
obj File is type:FileType and name:FileName and data:FileData;
obj FileType is a:ASCIIType or b:BinaryType or rolo:RolodexType;
obj FileName is Name;
obj FileData is ascii:ASCIIData or bin:BinaryData or rolo:RolodexData;
obj ASCIIData is string*;
obj BinaryData is number*;
obj RolodexData is Rolodex;
obj ASCIIType;
obj BinaryType;
obj RolodexType;


(**
 * Operations New, Open, Save, SaveAs, and Print are the basic file management
 * opertions.
 *)

operation New()->r':Rolodex
    post: r' = nil;
end New;

operation Open(fs:FileSpace, n:Name)->r':Rolodex
    post:
        (*
         * There is a file of the given name in either ASCII or Rolodex format
         * in the given file space.  If the latter, then the cards of the
         * output rolodex contain the converted ASCII data.  If the former
         * (i.e., file is in Rolodex format), then the cards of the output are
         * those in the file.
         *)
        exists (f in fs)
            (* File and rolodex names are the same *)
            (r'.name = f.name)

                and

            (* File data are the same as rolodex cards, parsed if ASCII *)
            if (f.data?ascii) then
                r'.cards = ParseRolodexText(f.data.ascii)
            else if (f.data?rolo) then
                r' = f.data.rolo
            else false; (* Neither ascii nor Rolodex file format *)

    description: (*
        Open is essentially a find operation, with the same underlying form of
        postcondition as a basic find.  This more concrete version of Open must
        parse ASCII text data files, if that is format in which a rolodex was
        saved.

        The postcondition relies on a non-trivial auxiliary function,
        ParseRolodexText, that would specify precisely the correspondence
        between the textual and abstract representations of rolodex card data.
    *);
end Open;

operation Save(fs:FileSpace, r:Rolodex)->fs':FileSpace
    description: (*
        Save is essentially an add operation.  It should be foramally specified
        in a comparable manner.
    *);
end Save;

operation Save(fs:FileSpace, r:Rolodex, type:FileType)->fs':FileSpace
    description: (*
        This version of Save operates the same as the preceding more abstract
        version, except that it uses the FileType input to determine the format
        of the file in the output filespace.
    *);

end Save;
```

```
operation SaveAs(fs: FileSpace, r:Rolodex, n:Name)->fs':FileSpace

    post: ;  (* Comparable change to Save. *)

end SaveAs;

operation Print(r:Rolodex)->rp:RolodexPrintout
    post:
        exists (r':Rolodex, c:Card) (
            ((c in r'.cards) iff (c in r.cards)) and
            SortedByName(r'.cards) and
            (rp = GeneratePrint(r'))
        );

    description: (*
        Printing produces the ASCII text for a sorted Rolodex.  The
        postcondition here relies on a non-trivial auxiliary function,
        GeneratePrintText, that would define textual formatting details
        precisely.  GeneratePrintText is the complement of the earlier
        ParseRolodexText function.

        Note that the postcondition specifies sorting on the rolodex, rather
        than on the printout, as would be the case with the following logic:

            forall (c in r)
                (GenerateCardPrint(c) in rp) and SortedByName(rp)

        This logic for sorting a printout would be considerably more tedious,
        since it would have to perform parsing to extract the name field from
        the text representation of a card.
    *);

end Print;

obj RolodexPrintout;

(**
 * The following auxiliary functions should be defined as discussed in the
 * formal spec primer
 *)
function ParseRolodexText(ASCIIData)->Card*;
function GeneratePrint(Rolodex)->RolodexPrintout;
function SortedByName(Card*)->boolean;


(**
 * Object UserTable and related objects define data needed for the
 * specification of security.
 *)
obj UserTable is UserInfo*;
obj UserInfo is uid:UserId and level:Level;
obj UserId is string;
obj Level is priv:Privileged or nonpriv:Nonprivileged;
obj Privileged;
obj Nonprivileged;

(**
 * Operations Login and FindUser specify a rudimentary form of security.
 *)
op Login(uid:UserId,state:SystemState)->(r':Rolodex)
    post:
        (* Adds are OK in the output Rolodex if the given user
         * is privileged, otherwise adds are not OK. *)
        if FindUser(state.users,uid).level?priv
        then r'.state.addok = true
        else r'.state.addok = false

            and

        (* The output Rolodex has no cards. *)
        (r'.cards = nil);

    description: (*
        The operational model here is that the login operation creates an
        initial empty Rolodex, with the appropriate value for the addok flag.
        This model affects the specification of the File Open operation, in
        that the Open will read Rolodex cards from a file, but maintain the
        Rolodex state established by Login. *);
end;

op FindUser(ut:UserTable,uid:UserId)->(uinfo':UserInfo)
    post:
        (* The output user info is that info in the given user table
         * with the given user id. *)
        forall (uinfo in ut)
            if (uinfo.uid = uid) then uinfo' = uinfo else uinfo' != uinfo;
end;


(**
 * Object Options and its subsidiaries define data needed for checkpointing.
 * The last clause in the Add operation operation above specifies the
 * checkpointing formally.
 *)
obj Options is chkpton:CheckpointOnOff and chkptinterval:CheckpointInterval
    and showprevdata:ShowPreviousData;
obj CheckpointCount is integer;
obj CheckpointInterval is integer;
obj CheckpointOnOff is boolean;


(**
 * Object CardData and related objects define data used in the historical
 * dialog opertions defined below.
 *)
obj CardData is c:Card and flag:boolean;
object LastAddInput is Card;
object LastDeleteInput is Name;
object LastChangeInput is Name and Card;
object LastFindInput is Name;
object ShowPreviousData is boolean;
```

```
(**
 * Operations InitiateAdd and ConfirmAdd specify a rudimentary form of
 * historical dialog managment for the Add operation.  Historical dialogs could
 * be similarly specified for other Rolodex operations.
 *)

operation InitiateAdd(r:Rolodex)->(cd:CardData)
    post:
        (* If the historical dialog option is on, then output the previously
         * entered card data, else output an empty card data. *)
        if r.state.options.showprevdata
        then cd.c = r.state.lastadd
        else cd.c = nil;
end;

operation ConfirmAdd(r:Rolodex, cd:CardData)->(r':Rolodex)
    pre: (* Same precondition as original Add,  but replace all occurrences of
          * variable c with cd.c *);
    post: (* Same postcondition as original Add, with same replacement as in
           * precondition *);
end;


(**
 * Object SystemState holds various compoinents of the system state, used by
 * the indicated operations defined above.
 *)
object SystemState is
    prev:Card*                          (* For Undo *)
        and
    users:UserTable and addok:boolean   (* For security *)
        and
    chkpt:integer                       (* For checkpointing *)
        and
    lastadd:LastAddInput and            (* For historical dialgos *)
    lastdel:LastDeleteInput and
    lastchange:LastChangeInput and
    lastfind:LastFindInput and
        and
    options:Options;                    (* For checkpointing and historical *)
```