with Ada.Integer\_Text\_IO;           use Ada.Integer\_Text\_IO;

package body Arbre\_Genealogique is

    procedure Afficher\_Ensemble2 is new Arbre\_Binaire\_Character.Piles\_Cle.Afficher\_Pile(Afficher\_Element => Afficher\_Entier);

                                    --TESTS--

    procedure Init\_AG(*Cle*: in Integer; *AG*: out T\_Branch) is

        begin

            Initialiser(Cle,AG);

        end Init\_AG;

    function Est\_Nul\_AG(*AG*:in T\_Branch) return Boolean is

        begin

            return Est\_Nul(AG);

        end Est\_Nul\_AG;

    function Est\_Present(*Cle*: in Integer; *AG*: in T\_Branch) return Boolean is

        begin

            if not Est\_Nul(AG) then

                return (not Est\_Nul(Rech\_Noeud(Cle,AG)));

            else

                return False;

            end if;

        end Est\_Present;

    procedure Init\_Foret(*F\_Foret*: in out Foret.T\_Pile) is

        choix:Character;

        Cle:Integer;

        AG:T\_Branch;

        begin

            Foret.Initialiser(F\_Foret); --Initialisation de la pile de forêt

            Put("Saisissez la clé par laquelle vous voulez initialiser le premier arbre de la forêt : "); Get(Cle);New\_Line;

            while Cle=-181199 or Cle=-34404 loop

                Put("Les valeurs -181199 et -34404 sont utilisées intérieurement pour assurer le fonctionnement de l'application. Veuillez saisir une autre valeur : "); Get(Cle);New\_Line;

            end loop;

            loop

                Put("Voulez-vous ajouter un autre arbre généalogique à la forêt? [y/n] : ");Get(choix);New\_Line;

                    if choix/='n' and choix/='N' then

                        Put("Saisissez la clé par laquelle vous voulez initialiser cet arbre : "); Get(Cle);New\_Line;

                        while Cle=-181199 or Cle=-34404 loop

                            Put("Les valeurs -181199 et -34404 sont utilisées intérieurement pour assurer le fonctionnement de l'application. Veuillez saisir une autre valeur : "); Get(Cle);New\_Line;

                        end loop;

                        New\_Line;

                        Init\_AG(Cle,AG); --Initialisation de l'arbre généalogique par la clé Cle

                        Foret.Empiler(F\_Foret, AG); --Empiler l'AG

                    end if;

                exit when choix='n' or choix ='N'; --Le programme s'arrête quand l'utilisateur a ajouté autant d'AG qu'il veuille.

            end loop;

        end Init\_Foret;

                        --FONCTIONS/PROCEDURES GENERATIONNELLES-

    function Nombre\_Ancetres(*Descendant*: in Integer; *AG*: in T\_Branch) return Integer is

        begin

            return 1 + Nbr\_Fils\_Noeud(Descendant,AG);

        end Nombre\_Ancetres;

    procedure Ensemble\_Ancetres\_Noeud(*Descendant*: in Integer; *AG*: in T\_Branch) is

        Ens:Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile;

        begin

            if Est\_Nul\_AG(AG) then

                Put\_Line("Impossible de chercher. L'arbre est vide!");

            else

                Ens:=Ensemble\_Fils\_Noeud(Descendant,AG);New\_Line;

                if Arbre\_Binaire\_Character.Piles\_Cle.Est\_Vide(Ens) then

                    Put\_Line("Cet individu n'a aucun parent connu.");New\_Line;

                else

                    Put\_Line("Les ancêtres de la clé" & Integer'Image(Descendant) & " sont : ");

                    Afficher\_Ensemble2(Ens);New\_Line;New\_Line;

                end if;

            end if;

        end Ensemble\_Ancetres\_Noeud;

    procedure Ensemble\_Ancetres\_Meme\_Generation(*g,Descendant*:in Integer; *AG*: in T\_Branch) is

        begin

            if g<0 then

                Put\_Line("La génération saisie est invalide!");

            elsif g=0 then

                Put\_Line("L'ancêtre de génération 0 par rapport au descendant" & Integer'Image(Descendant) & " est lui-même!");

            else

                Put("Les ancêtres de génération" & Integer'Image(g) & " par rapport à" & Integer'Image(Descendant) & " sont: ");

                Ensemble\_Meme\_Generation(g,Descendant,AG); New\_Line;

            end if;

        end Ensemble\_Ancetres\_Meme\_Generation;

    procedure Ensemble\_Ancetres\_Generation\_N(*g,Descendant*: in Integer; *AG*: in T\_Branch) is

        begin

            if g<0 then

                Put\_Line("La génération saisie est invalide!");

            elsif g=0 then

                Put\_Line("L'ancêtre de génération 0 par rapport au descendant" & Integer'Image(Descendant) & " est lui-même!");

            else

                Put\_Line("Les ancêtres de génération" & Integer'Image(g) & " ou moins par rapport au descendant" & Integer'Image(Descendant) & " sont:");

                Ensemble\_n\_Generation(g,Descendant,AG);New\_Line;

            end if;

        end Ensemble\_Ancetres\_Generation\_N;

    procedure Ensemble\_Un\_Parent(*AG*: in T\_Branch) is

        Ens: Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile;

        begin

            if not Est\_Nul(AG) then

                Ens:=Ensemble\_Un\_Fils(AG);New\_Line;

                if Arbre\_Binaire\_Character.Piles\_Cle.Est\_Vide(Ens) then

                    Put\_Line("Il n'y a aucun individu dont un seul parent est connu.");New\_Line;

                else

                    Put("Les individus ayant un seul parent connu sont:");

                    Afficher\_Ensemble2(Ens);New\_Line;New\_Line;

                end if;

            else

                Put\_Line("Impossible de chercher. L'arbre est vide.");

            end if;

        end Ensemble\_Un\_Parent;

    procedure Ensemble\_Deux\_Parents(*AG*: in T\_Branch) is

        Ens: Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile;

        begin

            if not Est\_Nul(AG) then

                Ens:=Ensemble\_Deux\_Fils(AG);

                if Arbre\_Binaire\_Character.Piles\_Cle.Est\_Vide(Ens) then

                    Put\_Line("Il n'y a aucun individu dont les deux parents sont connus.");New\_Line;

                else

                    Put("Les individus ayant les deux parents connus sont:");

                    Afficher\_Ensemble2(Ens);New\_Line;New\_Line;

                end if;

            else

                Put\_Line("Impossible de chercher. L'arbre est vide.");

            end if;

        end Ensemble\_Deux\_Parents;

    procedure Ensemble\_Orphelins(*AG*: in T\_Branch) is

        Ens: Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile;

        begin

            if not Est\_Nul(AG) then

                Ens:=Ensemble\_Feuilles(AG);

                Put("Les individus n'ayant aucun parent connu sont:");

                Afficher\_Ensemble2(Ens);New\_Line;New\_Line;

            else

                Put\_Line("Impossible de chercher. L'arbre est vide.");

            end if;

        end Ensemble\_Orphelins;

                        --FONCTIONS/PROCEDURES DE RECHERCHE--

    function Rech\_Noeud\_AG(*Cle*: in Integer; *AG*: in T\_Branch) return T\_Branch is

        begin

            return Rech\_Noeud(Cle,AG);

        end Rech\_Noeud\_AG;

    procedure Affecter\_Rech\_Noeud\_AG(*Cle*: in Integer;*AG*: in T\_Branch; *Noeud*: in out T\_Branch) is

        begin

            Affecter\_Rech\_Noeud(Cle,AG,Noeud);

        end Affecter\_Rech\_Noeud\_AG;

    function Access\_Tree\_Forest(*l*:in Integer; *F\_Foret*: in Foret.T\_Pile) return T\_Branch is

        Foret\_V:Foret.T\_Pile;

        begin

            if l=Foret.Size\_Pile(F\_Foret) then

                return Sommet(F\_Foret);

            else

                Affecter\_Pile(Foret\_V,F\_Foret);

                for i in 1..(Size\_Pile(F\_Foret)-l) loop

                    Affecter\_Pile(Foret\_V,Next\_Pile(Foret\_V));

                end loop;

                return Sommet(Foret\_V);

            end if;

        end Access\_Tree\_Forest;

    function Descendants\_Noeud\_Foret(*Cle*: in Integer; *F\_Foret*: in Foret.T\_Pile) return Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile is

        Parcours: Foret.T\_Pile;

        Recherche: T\_Branch;

        Cle\_Desc:Integer;

        Ens\_Noeud: Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile;

        begin

            if Cle=-34404 or Foret.Est\_Vide(F\_Foret) then --Si l'exception Arbre\_Vide a été soulevé dans Half\_Sibling\_Foret ou si la forêt est vide

                Arbre\_Binaire\_Character.Piles\_Cle.Initialiser(Ens\_Noeud);

                return Ens\_Noeud; --retourner un ensemble vide

            else

                Initialiser(Parcours);

                Affecter\_Pile(Parcours,F\_Foret); --Notre parcours dans la forêt

                Arbre\_Binaire\_Character.Piles\_Cle.Initialiser(Ens\_Noeud); --L'éventuel ensemble de descendants de la clé Cle

                while (not Foret.Est\_Vide(Parcours)) loop --Tant que la pile d'arbres n'est pas vide

                    Affecter\_Rech\_Noeud\_AG(Cle,Foret.Sommet(Parcours),Recherche); --Recherche reçoit le noeud contenant la clé Cle dans l'arbre sommet de la pile F\_Foret

                    if not Est\_Nul\_AG(Recherche) then --Si on a trouvé un noeud de clé Cle dans cet arbre

                        Cle\_Desc:=Nodekey(Rech\_Ancetre(Nodekey(Recherche),Foret.Sommet(Parcours))); --Cle\_Desc reçoit le descendant de cette clé

                        if Cle\_Desc/=-34404 then --Si aucune erreur n'a été soulevée

                            Arbre\_Binaire\_Character.Piles\_Cle.Empiler(Ens\_Noeud,Cle\_Desc); --On ajoute ce descendant dans l'ensemble

                        end if;

                    end if;

                    Affecter\_Pile(Parcours,Next\_Pile(Parcours)); --On dépile Parcours pour parcourir tous les arbres de la forêt

                end loop;

                return Ens\_Noeud; --Retourner l'ensemble des clés des descendants de l'individu Cle

            end if;

        end Descendants\_Noeud\_Foret;

    procedure Afficher\_Descendants\_Noeud\_Foret(*Cle*: in Integer; *F\_Foret*: in Foret.T\_Pile) is

        begin

            Afficher\_Ensemble2(Descendants\_Noeud\_Foret(Cle, F\_Foret)); --Affichage des clés

                    New\_Line;

        end Afficher\_Descendants\_Noeud\_Foret;

    procedure Half\_Sibling\_Foret(*Cle*: in Integer; *AG*: in T\_Branch; *F\_Foret*: in Foret.T\_Pile) is

        function Half\_Siblings(*Cle*: in Integer; *AG*: in T\_Branch; *F\_Foret*: in Foret.T\_Pile) return Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile is

            Noeud\_Cle: constant T\_Branch:=Rech\_Noeud\_AG(Cle,AG);

            Desc\_Pere,Desc\_Mere: Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile;

            begin

                if not Est\_Nul\_AG(Noeud\_Cle) then --Si la clé Cle existe

                    Arbre\_Binaire\_Character.Piles\_Cle.Affecter\_Pile(Desc\_Mere,Descendants\_Noeud\_Foret(Nodekey(Fils\_Droit(Noeud\_Cle)),F\_Foret)); --Desc\_Mere reçoit l'ensemble des descendants de la "mère" qui existent dans toute la forêt

                    Arbre\_Binaire\_Character.Piles\_Cle.Affecter\_Pile(Desc\_Pere,Descendants\_Noeud\_Foret(Nodekey(Fils\_Gauche(Noeud\_Cle)),F\_Foret));--Desc\_Pere reçoit l'ensemble des descendants du "père" qui existent dans toute la forêt

                    Arbre\_Binaire\_Character.Piles\_Cle.Supprimer\_Element(Cle,Desc\_Pere); --Suppression de l'individu Cle de l'ensemble des enfants du Père

                    Arbre\_Binaire\_Character.Piles\_Cle.Supprimer\_Element(Cle,Desc\_Mere); --Suppression de l'individu Cle de l'ensemble des enfants de la Mère

                    --Concaténation des deux ensembles

                    while (not Arbre\_Binaire\_Character.Piles\_Cle.Est\_Vide(Desc\_Mere)) loop --Tant que l'ensemble Desc\_Mere n'est pas vide

                        if not Arbre\_Binaire\_Character.Piles\_Cle.Existe\_Pile(Arbre\_Binaire\_Character.Piles\_Cle.Sommet(Desc\_Mere),Desc\_Pere) then --Si l'élément n'existe pas déjà dans Desc\_Pere alors

                            Arbre\_Binaire\_Character.Piles\_Cle.Empiler(Desc\_Pere,Arbre\_Binaire\_Character.Piles\_Cle.Sommet(Desc\_Mere)); --On empile le sommet de Desc\_Mere dans Desc\_Pere

                        else --Si le sommet existe déjà dans Desc\_Père et Desc\_Mère c'est qu'il sagit soit d'un frère soit d'une soeur. Il faut alors le supprimer de la pile.

                            Arbre\_Binaire\_Character.Piles\_Cle.Supprimer\_Element(Arbre\_Binaire\_Character.Piles\_Cle.Sommet(Desc\_Mere),Desc\_Pere); --Suppression de cet enfant commun de la pile Desc\_Pere.

                        end if;

                        Arbre\_Binaire\_Character.Piles\_Cle.Depiler(Desc\_Mere);

                    end loop;

                    return Desc\_Pere; --Retourner la concaténation des deux ensembles, ou Desc\_Pere si Desc\_Mere est vide ou une pile vide si les deux sont vides.

                else --Si la clé Cle existe

                    Arbre\_Binaire\_Character.Piles\_Cle.Initialiser(Desc\_Pere); --Pile vide

                    return Desc\_Pere; --Retrouner une pile vide

                end if;

            end Half\_Siblings;

        Ens:Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile;

        begin

            if Foret.Est\_Vide(F\_Foret) or Est\_Nul\_AG(AG) then --Si la forêt ou l'arbre sont vides

                Put\_Line("Impossible de chercher! L'arbre ou la forêt sont vides."); New\_Line;

            else

                Arbre\_Binaire\_Character.Piles\_Cle.Initialiser(Ens); --Initialisation de l'ensemble de clés

                Arbre\_Binaire\_Character.Piles\_Cle.Affecter\_Pile(Ens,Half\_Siblings(Cle,AG,F\_Foret)); --Ens reçoit l'ensemble des clés ayant au moins un parent en commun

                if not Arbre\_Binaire\_Character.Piles\_Cle.Est\_Vide(Ens) then --Si la clé a bien des demi-soeurs ou demi-frères alors

                    Put("Les demi-frères et demi-soeurs de la clé" & Integer'Image(Cle) & " sont : ");

                    Afficher\_Ensemble2(Ens); --Affichage des clés

                    New\_Line;

                    New\_Line;

                else

                    Put\_Line("Cette clé n'a ni des demi-soeurs ni des demi-frères");New\_Line;

                end if;

            end if;

        end Half\_Sibling\_Foret;

        procedure Multiplier\_10\_Foret(*F\_Foret*: in out Foret.T\_Pile) is

            Foret\_V:Foret.T\_Pile;

            AGForet: T\_Branch;

            begin

                if Est\_Vide(F\_Foret) then

                    Put\_Line("La forêt est vide!"); New\_Line;

                else

                    Affecter\_Pile(Foret\_V,F\_Foret);

                    while not Est\_Vide(Foret\_V) loop

                        AGForet:=Sommet(Foret\_V);

                        Multiplier\_10(AGForet);

                        Affecter\_Pile(Foret\_V,Next\_Pile(Foret\_V));

                    end loop;

                end if;

            end Multiplier\_10\_Foret;

                                        --AJOUT--

    function New\_Key\_Interval(*Cle*: in Integer; *Predecesseur*: in Integer; *AG*: in T\_Branch) return Arbre\_Binaire\_Character.Piles\_Cle.T\_Pile is

        begin

            return NewKeyInterval(Cle,Predecesseur,AG);

        end New\_Key\_Interval;

    procedure Ajouter\_Ancetre(*Ancetre*: in Integer; *Sexe*: in Character; *Descendant*: in Integer; *AG*: in out T\_Branch) is

        begin

            Ajouter2(Ancetre,Sexe,Descendant,AG);

        end Ajouter\_Ancetre;

                                 --MODIFICATION--

    procedure Modifier\_Cle\_Racine\_AG(*Cle*: in Integer; *AG*: in out T\_Branch) is

        begin

            Modifier\_Cle\_Racine(Cle,AG);

        end Modifier\_Cle\_Racine\_AG;

    procedure Modifier\_Cle\_AG(*Ancetre,NewAncetre*: in Integer; *AG*: in out T\_Branch) is

        begin

            Modifier\_Cle(Ancetre,NewAncetre,AG);

        end Modifier\_Cle\_AG;

    procedure Modifier\_Sexe\_AG(*Ancetre*: in Integer;*NewSexe*:in Character; *AG*: in out T\_Branch) is

        begin

            Modifier\_Donnee(Ancetre,NewSexe,AG);

        end Modifier\_Sexe\_AG;

                              --SUPPRESSION--

    procedure Supprimer\_Famille(*Descendant*: in Integer; *AG*: in out T\_Branch) is

        begin

            Supprimer\_Cle\_ET\_Fils(Descendant,AG);

        end Supprimer\_Famille;

                              --AFFICHAGE--

    procedure Afficher\_AG\_A\_Partir(*Descendant*: in Integer; *AG*: in T\_Branch) is

        begin

            Afficher\_A\_Partir(Descendant,AG);

        end Afficher\_AG\_A\_Partir;

    procedure Afficher\_AG(*AG*: in T\_Branch) is

        Depth\_AG: constant Integer :=Depth(AG);

        begin

            Put(" 0       ");

            for i in 1..(Depth\_AG-1) loop

                Put(Integer'Image(i));

                Put("  ");

            end loop;

            Put("   Générations");New\_Line;

            Put("-");

            for i in 1..Depth\_AG loop

                Put("----");

            end loop;

            Put("-------------------");

            New\_Line;

            Afficher(AG);

        end Afficher\_AG;

end Arbre\_Genealogique;