with Ada.Text\_IO;                   use Ada.Text\_IO;

with piles;

with arbre\_binaire;

package Arbre\_Genealogique is

    --Instantiation du module ABR.

    package Arbre\_Binaire\_Character is

        new arbre\_binaire (T\_Value => Character, Zero => 'E');

    use Arbre\_Binaire\_Character;

    package Foret is

        new Piles (T\_Element => T\_Branch);

    use Foret;

    procedure Afficher is new Afficher\_ABR (Afficher\_Donnee => Put);

    procedure Afficher\_A\_Partir is new Afficher\_APartir(Afficher\_Donnee => Put);

                                --TESTS--

        --procedure Initialiser (Cle: in Integer; Arbre : out T\_Branch);

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

        --Initialiser la forêt d'arbres généalogiques

    procedure Init\_Foret(*F\_Foret*: in out Foret.T\_Pile); --####SECONDE PARTIE####

        --Vérifie si l'arbre est nul.

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

        --Vérifie si une clé est présente dans l'abre.

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

                    --FONCTIONS/PROCEDURES GENERATIONNELLES-

        --function Nbr\_Fils\_Noeud(Cle: in Integer; Arbre: in T\_Branch) return Integer; +1 pour inclure le noeud lui-même!

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

        --function Ensemble\_Fils\_Noeud(Cle: in Integer; Arbre: in T\_Branch) return T\_Pile;

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

        --procedure Ensemble\_Meme\_Generation(g,Cle:in Integer; Arbre: in T\_Branch);  (Ensemble d'ancêtres de génération g par rapport au la génération du descendant!)

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

        --procedure Ensemble\_n\_Generation(g,Cle:in Integer; Arbre: in T\_Branch); (Ensemble d'ancêtres de génération g ou moins par rapport à la génération du descendant!)

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

    -- procedure Ensemble\_Un\_Fils(Arbre:in T\_Branch);

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

        --procedure Ensemble\_Deux\_Fils(Arbre:in T\_Branch);

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

        --procedure Ensemble\_Feuilles(Arbre:in T\_Branch);

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

                    --FONCTIONS/PROCEDURES DE RECHERCHE--

        --Récupérer la cellule d'un noeud à partir de sa clé.

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

        --Affecter à Noeud la cellule de clé Cle dans l'arbre (i.e., affecter le resultat de Rech\_Noeud\_AG(Cle,AG) à Noeud).

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

        --##################################################SECONDE PARTIE##################################################--

        --Récupérer le lème arbre d'une forêt

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

        --Récupérer tous les descendants d'une clé à partir de ses occurences dans la forêt.

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

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

        --Récupérer les demi-frères et demi-soeurs d'un individu

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

        --Multiplier toutes les clés de la FORET par 10

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

                                --AJOUT--

        --Récupérer l'intervalle (piles) de valeurs permises que peut prendre le fils éventuel d'un noeud en testant avec une clé quelconque.

        --par convention, la pile sera de la forme [max,0,-181199] pour représenter l'intervalle ]-INFINI,max], [-181199,0,min] pour représenter l'intervalle [min,+INFINI[ et [max,min] pour représenter l'intervalle [min,max].

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

        --procedure Ajouter2(Cle\_Nouveau\_Noeud: in Integer; Donnee\_Nouveau\_Noeud: in T\_Value; Cle\_Noeud\_Parent:in integer; Arbre: in out T\_Branch);

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

                            --MODIFICATION--

        --Modifier la clé de la racine ou du noeud poussé en argument.

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

        --procedure Modifier\_Cle(Cle,NewCle: in Integer; Arbre: in out T\_Branch);

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

        --procedure Modifier\_Donnee(Cle: in Integer; NewDonnee: in T\_Value; Arbre: in out T\_Branch);

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

                            --SUPPRESSION--

        --procedure Supprimer\_Cle\_ET\_Fils(Cle: in Integer; Arbre: in out T\_Branch);

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

                             --AFFICHAGE--

        --procedure Afficher\_APartir(Cle:in Integer; Arbre: in T\_Branch);

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

        --procedure Afficher\_ABR(Arbre: in T\_Branch);

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

end Arbre\_Genealogique;