```
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 I
nteger ;
        --
function Ensemble_Fils_Noeud(Cle: in Integer; Arbre: in T_Branch) return T_Pil
e;
    procedure Ensemble_Ancetres_Noeud(Descendant: in Integer; AG: in T_Branch)
;
        --
procedure Ensemble_Meme_Generation(g,Cle:in Integer; Arbre: in T_Branch);  (En
semble d'ancêtres de génération g par rapport au la génération du descendant!)
```

```
    procedure Ensemble_Ancetres_Meme_Generation(g,Descendant:in Integer; AG: i
n T_Branch);

        --
procedure Ensemble_n_Generation(g,Cle:in Integer; Arbre: in T_Branch); (Ensemb
le d'ancêtres de génération g ou moins par rapport à la génération du descenda
nt!)
    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 resulta
t de Rech_Noeud_AG(Cle,AG) à Noeud).
    procedure Affecter_Rech_Noeud_AG(Cle: in Integer;AG: in T_Branch; Noeud: i
n 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 fo
rêt.
    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 Fo
ret.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 F
oret.T_Pile); --####SECONDE PARTIE####
        --Multiplier toutes les clés de la FORET par 10
    procedure Multiplier_10_Foret(F_Foret: in out Foret.T_Pile);
```

```ada
                                --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;
```