```
with Ada.Text_IO;                use Ada.Text_IO;
with Ada.Integer_Text_IO;        use Ada.Integer_Text_IO;
with Ada.Unchecked_Deallocation;

package body Arbre_Binaire is

        procedure Free is
            new Ada.Unchecked_Deallocation (T_Node, T_Branch);


        procedure Afficher_Entier(Element: in Integer) is
            begin
                Put(Integer'Image(Element));
            end Afficher_Entier;
        procedure Afficher_Ensemble is new Piles_Cle.Afficher_Pile (Afficher_Element => Aff
icher_Entier);

                            --TESTS--

    procedure Initialiser_Vide(Arbre: out T_Branch) is
        begin
            Arbre:=Null;
        end Initialiser_Vide;

    procedure Initialiser (Cle: in Integer; Arbre : out T_Branch)is
        begin
            Arbre:=New T_Node;
            Arbre.all.Cle:= Cle;
        end Initialiser;

    function Est_Nul (Arbre : in T_Branch) return Boolean is
        begin
            return Arbre=Null;
        end Est_Nul;

    function Gauche_ou_Droite(Cle:in Integer;Arbre: in T_Branch) return Character is
            begin
                if not Est_Nul(Rech_Noeud(Cle,Arbre)) then --Si la clé existe dans l'arbre
                    if Est_Nul(Rech_Ancetre(Cle,Arbre)) then --
Si la clé n'a pas de parent, i.e., la clé est la racine
                        return 'R';
                    else
                        if Nodekey(Fils_Gauche(Rech_Ancetre(Cle,Arbre)))=Cle then -
- Si la clé est un fils gauche
                            return 'G';
                        else --Si la clé est un fils droit
                            return 'D';
                        end if;
```

```
                end if;
            else --Si la clé n'existe pas
                return 'E'; --'E' pour Erreur ou Empty
            end if;
        end Gauche_ou_Droite;


                --FONCTIONS ELEMENTAIRES--

function Nodekey(Arbre: in T_Branch) return Integer is
    begin
        if Arbre /= Null then
            return Arbre.all.Cle;
        else
            raise Cle_Absente_Exception;
        end if;
        exception
            when Cle_Absente_Exception =>
                Null;
                return -34404; --Leetspeak signifiant ERROR
    end Nodekey;

function NodeValue(Arbre: in T_Branch) return T_Value is
    begin
        if Arbre/=Null then
            return Arbre.all.Donnee;
        else
            raise Arbre_Vide;
        end if;
        exception
            when Arbre_Vide =>
                Null;
                return Zero;
    end NodeValue;

function Fils_Droit(Arbre: in T_Branch) return T_Branch is
    begin
        if Arbre/=Null then
            return Arbre.all.FilsD;
        else
            return Null;
        end if;
    end Fils_Droit;

function Fils_Gauche(Arbre: in T_Branch) return T_Branch is
    begin
        if Arbre/=Null then
            return Arbre.all.FilsG;
```

```
            else
                return Null;
            end if;
        end Fils_Gauche;

    procedure Multiplier_10(Arbre: in out T_Branch) is
        procedure Multiplier_10_fils(Arbre:in T_Branch) is
            begin
                if Arbre/=Null then
                    if Arbre.all.FilsG/=Null then
                        Arbre.all.FilsG.all.Cle:=Arbre.all.FilsG.all.Cle*10; --
Multiplier la clé du fils gauche par 10
                        Multiplier_10_fils(Arbre.all.FilsG);                 --
Récursivité avec le fils gauche
                    end if;
                    if Arbre.all.FilsD/=Null then
                        Arbre.all.FilsD.all.Cle:=Arbre.all.FilsD.all.Cle*10; --
Multiplier la clé du fils droit par 10
                        Multiplier_10_fils(Arbre.all.FilsD);                 --
Récursivité avec le fils droit
                    end if;
                else
                    Null;
                end if;
            end Multiplier_10_fils;
        begin
            if Arbre/=Null then
                Multiplier_10_fils(Arbre); --Multiplier les clés de tous les fils par 10
                Arbre.all.Cle:=Arbre.all.Cle*10; --Multiplier la clé de la racine par 10
            else
                Null;
            end if;
        end Multiplier_10;

    function Depth(Arbre: in T_Branch) return Integer is
        function max(n,m: in Integer) return Integer is
            begin
                if n=m then
                    return n;
                elsif n>m then
                    return n;
                else
                    return m;
                end if;
            end max;
        begin
            if Arbre=Null then
                return 0;
```

```
            else
                return 1 + max(Depth(Arbre.all.FilsG),Depth(Arbre.all.FilsD));
            end if;
        end Depth;

    procedure Affecter_Arbre(Arbre1: in out T_Branch; Arbre2: in T_Branch) is
        begin
            Arbre1:=Arbre2;
        end Affecter_Arbre;


              --FONCTIONS/PROCEDURES GENERATIONNELLES--

    function Gen(Cle: in Integer; Arbre:in T_Branch) return Integer is
        begin
            if Arbre=Null or Est_Nul(Rech_Noeud(Cle,Arbre)) then
                return -34404;
            else
                if Arbre.all.Cle=Cle then
                    return 0;
                elsif Arbre.all.filsG/=Null and then (not Est_Nul(Rech_Noeud(Cle,Arbre.all.
FilsG))) then
                --Si le fils gauche n'est pas nul et la clé existe dans son sous-arbre
                    return 1 + Gen(Cle, Arbre.all.FilsG);
                elsif Arbre.all.filsD/=Null and then (not Est_Nul(Rech_Noeud(Cle,Arbre.all.
FilsD))) then
                --Si le fils droit n'est pas nul et la clé existe dans son sous-arbre
                    return 1 + Gen(Cle,Arbre.all.FilsD);
                else
                    return 0;
                end if;
            end if;
        end Gen;

    function Nbr_Fils_Noeud(Cle: in Integer; Arbre: in T_Branch) return Integer is
        function Nbr_Fils(Arbre: in T_Branch) return Integer is
            begin
                if Arbre/=Null then
                    if Arbre.all.FilsD/=Null and Arbre.all.FilsG/=Null then --
Si les deux fils existent
                        return 2 + Nbr_Fils(Arbre.all.FilsG) + Nbr_Fils(Arbre.all.FilsD);
                    elsif Arbre.all.FilsD=Null and Arbre.all.FilsG/=Null then -
- Si seul le fils gauche existe
                        return 1 + Nbr_Fils(Arbre.all.FilsG);
                    elsif Arbre.all.FilsD/=Null and Arbre.all.FilsG=Null then --
Si seul le fils droit existe
                        return 1 + Nbr_Fils(Arbre.all.FilsD);
                    else
                        return 0;
```

```
                    end if;
                else
                    return 0;
                end if;
            end Nbr_Fils;

        begin
            return Nbr_Fils(Rech_Noeud(Cle,Arbre));
        end Nbr_Fils_Noeud;

    function Ensemble_Fils_Noeud(Cle: IN Integer;Arbre: in T_Branch) return T_Pile is
        Original: constant T_Branch:=Rech_Noeud(Cle,Arbre);
        Ens: T_Pile;
        procedure Intermediaire(Arbre: in T_Branch) is
            begin
                if Arbre/=Null then
                    if Arbre.all.FilsD/=Null and Arbre.all.FilsG/=Null then --
Si les deux fils existent
                        Empiler(Ens,Arbre.all.FilsG.all.Cle);
                        Empiler(Ens,Arbre.all.FilsD.all.Cle);
                        Intermediaire(Arbre.all.FilsG);
                        Intermediaire(Arbre.all.FilsD);
                    elsif Arbre.all.FilsD=Null and Arbre.all.FilsG/=Null then -
- Si seul le fils gauche existe
                        Empiler(Ens,Arbre.all.FilsG.all.Cle);
                        Intermediaire(Arbre.all.FilsG);
                    elsif Arbre.all.FilsD/=Null and Arbre.all.FilsG=Null then --
Si seul le fils droit existe
                        Empiler(Ens,Arbre.all.FilsD.all.Cle);
                        Intermediaire(Arbre.all.FilsD);
                    else
                        Null;
                    end if;
                else
                    Null;
                end if;
            end Intermediaire;
        begin
            if Arbre=Null then
                Piles_Cle.Initialiser(Ens);
                return Ens;
            else
                Piles_Cle.Initialiser(Ens);
                Intermediaire(Original);
                return Ens;
            end if;
        end Ensemble_Fils_Noeud;
```

```
    function Nbr_Meme_Generation(g: in integer; Arbre: in T_Branch) return Integer is
        Original: Constant T_Branch := Arbre; --
Pour garder l'arbre original dans la récursivité
        function Intermediaire(g:in Integer; Arbre: in T_Branch) return Integer is
            begin
                if Arbre/=Null and g>=0 then
                    if g=0 then
                        return 1; --C'est la racine
                    elsif Gen(Arbre.all.Cle,Original)+1=g then --
Si la génération du noeud est directement au dessus de g
                        if Arbre.all.FilsD/= Null AND Arbre.all.FilsG/=Null then --
Si les deux fils existent
                            return 2;
                        elsif Arbre.all.FilsD=Null and Arbre.all.FilsG/=Null then --
Si seul le fils gauche existe
                            return 1;
                        elsif Arbre.all.FilsG=Null and Arbre.all.FilsD/= Null then --
Si seul le fils droit existe
                            return 1;
                        else
                            return 0;
                        end if;
                    elsif Gen(Arbre.all.Cle,Original)+1 < g then --
Si g n'est pas directement en dessous du noeud
                        return Intermediaire(g,Arbre.all.FilsG) + Intermediaire(g,Arbre.all
.FilsD);
                    else
                        return 0;
                    end if;
                else
                    return 0;
                end if;
            end Intermediaire;
        begin
            if Arbre=Null then
                return 0;
            else
                return Intermediaire(g,Original);
            end if;
        end Nbr_Meme_Generation;

    procedure Ensemble_Meme_Generation(g,Cle:in Integer; Arbre: in T_Branch) is
        Original: constant T_Branch:=Rech_Noeud(Cle,Arbre); --
Pour commencer à partir du noeud de clé Cle
        Ens: T_Pile;
        procedure Intermediaire(g:in Integer;Arbre:in T_Branch) is
            begin
                if Arbre=Null then
```

```
                    Null;
                else
                    if g=0 then
                        Empiler(Ens,Arbre.all.Cle); --C'est la racine
                    elsif Arbre.all.FilsG/=Null then
                        if Gen(Arbre.all.FilsG.all.Cle,Original)=g then --
Si la gén du fils gauche est égale à g
                            Empiler(Ens,Arbre.all.FilsG.all.Cle); --Empiler ce dernier
                        elsif Gen(Arbre.all.FilsG.all.Cle,Original)<g then
                            Intermediaire(g,Arbre.all.FilsG);
                        else
                            Null;
                        end if;
                    else
                        Null;
                    end if;
                    if Arbre.all.FilsD/=Null then
                        if Gen(Arbre.all.FilsD.all.Cle,Original)=g then --
Si la gén du fils droit est égale à g
                            Empiler(Ens,Arbre.all.FilsD.all.Cle); --Empiler ce dernier
                        elsif Gen(Arbre.all.FilsD.all.Cle,Original)<g then
                            Intermediaire(g,Arbre.all.FilsD);
                        else
                            Null;
                        end if;
                    else
                        Null;
                    end if;
                end if;
            end Intermediaire;
        begin
            if Original=Null then
                Null;
            else
                Piles_Cle.Initialiser(Ens); --Initialiser la pile
                Intermediaire(g,Original);
                Afficher_Ensemble(Ens); --
Afficher la pile remplie des clés de génération g par rapport à la clé Cle
            end if;
        end Ensemble_Meme_Generation;

    procedure Ensemble_n_Generation(g,Cle:in Integer; Arbre: in T_Branch) is
        begin
            if Arbre/=Null and 1<g then
                for i in 1..g loop
                    Put("        -Ceux de génération" & Integer'Image(i) & " : ");
                    Ensemble_Meme_Generation(i,Cle,Arbre);New_Line; --
Afficher les clés de génération i par rapport à Cle
```

```
                end loop;
            else
                Null;
            end if;
        end Ensemble_n_Generation;

    function Ensemble_Un_Fils(Arbre:in T_Branch) return T_Pile is
        Original: constant T_Branch:=Arbre;
        Ens: T_Pile;
        procedure Intermediaire(Arbre:in T_Branch) is
            begin
                if Arbre=Null then
                    Null;
                else
                    if Arbre.all.FilsG/=Null and Arbre.all.FilsD/=Null then --
Si les deux fils existent
                        Intermediaire(Arbre.all.FilsG);
                        Intermediaire(Arbre.all.FilsD);
                    elsif Arbre.all.FilsG/=Null and Arbre.all.FilsD=Null then --
Si seul le fils gauche existe
                        Empiler(Ens,Arbre.all.Cle); --Empiler la clé du noeud
                        Intermediaire(Arbre.all.FilsG);
                    elsif Arbre.all.FilsD/=Null and Arbre.all.FilsG=Null then --
Si seul le fils droit existe
                        Empiler(Ens,Arbre.all.Cle); --Empiler la clé du noeud
                        Intermediaire(Arbre.all.FilsD);
                    else
                        Null;
                    end if;
                end if;
            end Intermediaire;
        begin
            if Arbre=Null then
                Piles_Cle.Initialiser(Ens);
                return Ens; --Retourner la pile vide
            else
                Piles_Cle.Initialiser(Ens);
                Intermediaire(Original);
                return Ens; --Retourner la pile remplie des clés n'ayant qu'un fils
            end if;
        end Ensemble_Un_Fils;

    function Ensemble_Deux_Fils(Arbre:in T_Branch) return T_Pile is
        Original: constant T_Branch:=Arbre;
        Ens: T_Pile;
        procedure Intermediaire(Arbre:in T_Branch) is
            begin
                if Arbre=Null then
```

```
                    Null;
                else
                    if Arbre.all.FilsG/=Null and Arbre.all.FilsD/=Null then --
Si les deux fils existent
                        Empiler(Ens,Arbre.all.Cle); --Empiler la clé du noeud
                        Intermediaire(Arbre.all.FilsG);
                        Intermediaire(Arbre.all.FilsD);
                    elsif Arbre.all.FilsG/=Null and Arbre.all.FilsD=Null then --
Si seul le fils gauche existe
                        Intermediaire(Arbre.all.FilsG);
                    elsif Arbre.all.FilsD/=Null and Arbre.all.FilsG=Null then --
Si seul le fils droit existe
                        Intermediaire(Arbre.all.FilsD);
                    else
                        Null;
                    end if;
                end if;
            end Intermediaire;
        begin
            if Arbre=Null then
                Piles_Cle.Initialiser(Ens);
                return Ens; --retourner la pile vide
            else
                Piles_Cle.Initialiser(Ens);
                Intermediaire(Original);
                return Ens; --retourner la pile remplie des clés ayant deux fils
            end if;
        end Ensemble_Deux_Fils;

    function Ensemble_Feuilles(Arbre:in T_Branch) return T_Pile is
        Original: constant T_Branch:=Arbre;
        Ens: T_Pile;
        procedure Intermediaire(Arbre:in T_Branch) is
            begin
                if Arbre=Null then
                    Null;
                elsif Arbre.all.FilsG=Null and Arbre.all.FilsD=Null then --
Si c'est une feuille
                    Empiler(Ens,Arbre.all.Cle);
                elsif Arbre.all.FilsG/=Null and Arbre.all.FilsD/=Null then -
- SI les deux fils existent
                    Intermediaire(Arbre.all.FilsG);
                    Intermediaire(Arbre.all.FilsD);
                elsif Arbre.all.FilsG/=Null and Arbre.all.FilsD=Null then --
Si seul le fils gauche existe
                    Intermediaire(Arbre.all.FilsG);
                elsif Arbre.all.FilsD/=Null and Arbre.all.FilsG=Null then --
Si seul le fils droit existe
```

```
                Intermediaire(Arbre.all.FilsD);
            else
                Null;
            end if;
        end Intermediaire;
    begin
        if Arbre=Null then
            Piles_Cle.Initialiser(Ens);
            return Ens; --Retourner la pile vide
        else
            Piles_Cle.Initialiser(Ens);
            Intermediaire(Original);
            return Ens; --Retourner la pile remplie des feuilles de l'arbre
        end if;
    end Ensemble_Feuilles;


        --FONCTIONS/PROCEDURES DE RECHERCHE--

function Rech_Noeud(Cle: in Integer; Arbre: in T_Branch) return T_Branch is
    function Recherche(Cle: in Integer; Arbre: in T_Branch) return T_Branch is
        begin
            if Arbre/= Null then
                if Arbre.all.Cle=Cle then --Si la clé est égale à celle du noeud
                    return Arbre;
                elsif Arbre.all.Cle<Cle then --Si la clé est supérieur à celle du noeud
                    return Recherche(Cle,Arbre.all.FilsD);
                elsif Arbre.all.Cle>Cle then --Si elle est inférieure
                    return Recherche(Cle,Arbre.all.FilsG);
                else
                    return Null;
                end if;
            else
                return Null;
            end if;
        end Recherche;
    begin
        if Arbre=Null then
            return Null;
        else
            return Recherche(Cle, Arbre);   --retourne le noeud ayant pour clé Cle
        end if;
    end Rech_Noeud;

function Rech_Ancetre(Cle: in Integer; Arbre: in T_Branch) return T_Branch is
    begin
        if Arbre/= Null then
            if Arbre.all.Cle=Cle then --Si c'est la racine
```

```
                    return Null;
                elsif Arbre.all.Cle<Cle then --Si la clé est supérieure à celle du noeud
                    if Arbre.all.FilsD/=Null and then Arbre.all.FilsD.all.Cle=Cle then --
Si la clé est le fils droit du noeud
                        return Arbre;
                    else
                        return Rech_Ancetre(Cle,Arbre.all.FilsD);
                    end if;
                elsif Arbre.all.Cle>Cle then
                    if Arbre.all.FilsG/=Null and then Arbre.all.FilsG.all.Cle=Cle then --
Si la clé est le fils gauche du noeud
                        return Arbre;
                    else
                        return Rech_Ancetre(Cle,Arbre.all.FilsG);
                    end if;
                else
                    return Null;
                end if;
            else
                return Null;
            end if;
        end Rech_Ancetre;

    procedure Affecter_Rech_Noeud(Cle: in Integer;Arbre: in T_Branch; Noeud: in out T_Branc
h) is
        begin
            Noeud:=Rech_Noeud(Cle,Arbre);
        end Affecter_Rech_Noeud;

    function Donnee_Noeud(Cle : in Integer ; Arbre : in T_Branch) return T_Value is
        begin
            if Arbre/= Null then
                if Rech_Noeud(Cle,Arbre)/= Null then --Si la clé existe alors...
                    return Rech_Noeud(Cle,Arbre).all.Donnee; --retourner sa donnée
                else
                    return Zero;
                end if;
            else
                raise Arbre_Vide;
            end if;
            exception
                when Arbre_Vide=> Put_Line("Arbre vide! Donnée éronnée retournée."); return
 Zero;
        end Donnee_Noeud;

    function Cle_Noeud(Cle:in Integer; Arbre:in T_Branch) return Integer is
        begin
            if Arbre/=Null then
```

```
                if Rech_Noeud(Cle,Arbre)/= Null then    --Si la clé existe alors...
                    return Rech_Noeud(Cle,Arbre).all.Cle; --la retourner
                else
                    return -34404;
                end if;
            else
                raise Cle_Absente_Exception;
            end if;
            exception
                when Cle_Absente_Exception => Put_Line("Arbre vide! Clé éronnée retournée!"
); return -34404;
        end Cle_Noeud;



                        --SUPPRESSION--

    procedure Supprimer_Fils (Arbre: in out T_Branch) is
        begin
            if Arbre/=Null then
                Supprimer_Fils(Arbre.all.FilsG);
                Supprimer_Fils(Arbre.all.FilsD);
                Arbre:=Null;
                Free(Arbre);
            end if;
        end Supprimer_Fils;

    procedure Supprimer_Cle_ET_Fils(Cle: in Integer; Arbre: in out T_Branch) is
       Noeud1: T_Branch:=Rech_Noeud(Cle,Arbre); --Noeud qu'on veut supprimer
       Noeud2: constant T_Branch:=Rech_Ancetre(Cle, Arbre); --Son père
        begin
            if Arbre/=Null then
                Supprimer_Fils(Noeud1); --Supprimer les fils du noeud
                if Noeud2/=Null then --
Si le noeud qu'on veut supprimer ne s'agit pas de la racine
                    if Noeud2.all.FilsG/=Null and then Noeud2.all.FilsG.all.Cle=Cle then --
Si le noeud est un fils gauche
                        Noeud2.all.FilsG:=Null;
                        Free(Noeud2.all.FilsG);
                    else --Si le noeud est un fils droit
                        Noeud2.all.FilsD:=Null;
                        Free(Noeud2.all.FilsD);
                    end if;
                else --Si le noeud est la racine
                    Arbre:=Null;
                end if;
            else
                Null;
            end if;
```

```
        end Supprimer_Cle_ET_Fils;

    procedure Detruire(Arbre: in out T_Branch) is
        begin
            Supprimer_Fils(Arbre); --Supprimer les fils de la racine
            Arbre:=Null;           --Supprimer la racine elle-même
            Free(Arbre);
        end Detruire;


                        --AJOUT--

    function NewKeyInterval(Cle,Parent: Integer; Arbre:T_Branch) return T_Pile is
        min,max:Integer;
        Temp:T_Branch;
        minf,M_inf:Boolean;
        Ens1,Vide:T_Pile;
        begin
            Initialiser(Vide);
            if Cle/=Parent then --Si la clé est différente de son parent
                if Arbre/=Null then --Si l'arbre n'est pas vide
                    if Arbre.all.Cle=Parent then --Si le parent est la racine
                        if Cle>Parent then --
Si on veut que la clé soit le fils droit de la racine
                            Empiler(Ens1,Parent+1);Empiler(Ens1,0);Empiler(Ens1,-181199); -
-Intervalle possible des valeurs que peut prendre la clé [Parent+1, +INFINI[
                            return Ens1;
                        else  --
Puisque différent de Parent et non strictement supérieure à Parent donc automatique stricte
ment inférieur à Parent! càd qu'on veut que la clé soit le fils gauche de la racine
                            Empiler(Ens1,-181199); Empiler(Ens1,0); Empiler(Ens1,Parent-
1); --Intervalle possible des valeurs que peut prendre la clé : ]-INFINI,Parent-1]
                            return Ens1;
                        end if;
                    elsif Cle<Parent then --
Si on veut que la clé soit le fils gauche de Parent
                        if not Est_Nul(Rech_Noeud(Parent,Arbre)) and then Fils_Gauche(Rech_
Ancetre(Parent,Arbre))=Rech_Noeud(Parent,Arbre) then
                            --Si le noeud Parent est le fils gauche de son propre parent
                            Temp:=Rech_Noeud(Parent,Arbre); --
Le Noeud contenant la clé Parent
                            --Pour que la nouvelle clé soit le fils gauche du noeud
                            max:=NodeKey(Temp)-1;
                            while Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre and then Temp/=F
ils_Droit(Rech_Ancetre(NodeKey(Temp),Arbre)) loop
                            --
Tant que Temp n'est pas arrivée à la racine et que ce n'est pas un fils droit
```

```
                                Temp:=Rech_Ancetre(NodeKey(Temp),Arbre); --
Temp recoît son prédécesseur
                            end loop;
                            --
Temp est maintenant le premier noeud étant un fils droit parmi les prédécesseurs de Parent
                            if Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre then --
S'il ne s'agit pas de la racine
                                min:=NodeKey(Rech_Ancetre(NodeKey(Temp),Arbre))+1; --
La valeur minimale est celle de la clé du prédécesseur du premier prédécesseur étant un fil
s droit +1
                            else --Si c'est la racine
                                if not Est_Nul(Rech_Noeud(Parent,Fils_Gauche(Arbre))) then
--Si Parent existe dans le sous-arbre gauche de la racine
                                    minf:=True; --
Les valeurs possibles ne sont pas minorées
                                else --Si Parent existe dans le sous-
arbre droit de la racine, c'est que les valeurs possibles admettent forcément un minimum qu
i est la clé de la racine +1
                                    min:=NodeKey(Arbre)+1;
                                end if;
                            end if;
                            if minf then --Si les valeurs possibles ne sont pas minorées
                                Empiler(Ens1,-181199);Empiler(Ens1,0); --
intervalle du type ]-INFINI,max]
                            else
                                Empiler(Ens1,min); --intervalle du type [min,max]
                            end if;
                            Empiler(Ens1, max);
                            return Ens1;
                        elsif not Est_Nul(Rech_Noeud(Parent,Arbre)) and then Fils_Droit(Rec
h_Ancetre(Parent,Arbre))=Rech_Noeud(Parent,Arbre) then
                            --
Si le noeud Parent est le fils droit de son propre parent
                            Temp:=Rech_Noeud(Parent,Arbre); --
Le Noeud contenant la clé Parent

                            --Pour que la nouvelle clé soit le fils gauche du noeud
                            minf:=False;M_inf:=False;
                            max:=NodeKey(Temp)-1; --
Le maximum est automatiquement la clé du parent -1
                            while Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre and then Temp/=F
ils_Droit(Rech_Ancetre(NodeKey(Temp),Arbre)) loop --
Tant que le prédécesseur de Temp n'est pas la racine et que Temp n'est pas un fils droit
                                Temp:=Rech_Ancetre(NodeKey(Temp),Arbre); --
Temp recçoit son prédécesseur
                            end loop;
                            if Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre then --
Si le prédécesseur de Temp n'est pas la racine
```

```
                                min:=NodeKey(Rech_Ancetre(NodeKey(Temp),Arbre))+1; --
On a trouvé notre valeur minimale
                            else --Si c'est la racine
                                if not Est_Nul(Rech_Noeud(Parent,Fils_Gauche(Arbre))) then
--Si le Parent est dans le sours-arbre gauche de la racine
                                    min:=NodeKey(Temp)+1; --
La valeur minimale est celle de la clé du premier prédécesseur étant un fils droit +1
                                else --Si Parent est dans le sous-arbre doit de la racine
                                    min:=NodeKey(Arbre)+1; --
Ce sera automatiquement la valeur de la racine +1
                                end if;
                            end if;
                            --Intervalle du type [min,max]
                            Empiler(Ens1,min);
                            Empiler(Ens1,max);
                            --Intervalle du type [min,max]
                            return Ens1;
                        else --
Si Parent est ni un fils gauche ni un fils droit ni une racine (impossible)
                            return Vide;
                        end if;
                    else  --
Puisque différent et non strictement supérieure donc automatique strictement inférieur!
                        if not Est_Nul(Rech_Noeud(Parent,Arbre)) and then Fils_Gauche(Rech_
Ancetre(Parent,Arbre))=Rech_Noeud(Parent,Arbre) then
                            --
Si le noeud Parent est le fils gauche de son propre parent
                            --Pour que la nouvelle clé soit le fils droit du noeud
                            Temp:=Rech_Noeud(Parent,Arbre);
                            minf:=False;M_inf:=False;
                            min:=NodeKey(Temp)+1; --
La valeur minimale est automatique la clé du parent +1
                            while Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre and then Temp/=F
ils_Gauche(Rech_Ancetre(NodeKey(Temp),Arbre)) loop --
Tant que le prédécesseur de Temp est différent de la racine et que Temp n'est pas un fils g
auche
                                Temp:=Rech_Ancetre(NodeKey(Temp),Arbre);
                            end loop;
                            if Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre then --
Si le prédécesseur de Temp n'est pas la racine
                                max:=NodeKey(Rech_Ancetre(NodeKey(Temp),Arbre))-1; --
La valeur maximale est celle de la clé du prédécesseur du premier prédécesseur étant un fil
s gauche
                            else --Si le prédécesseur de temp est la racine
                                if not Est_Nul(Rech_Noeud(Parent,Fils_Gauche(Arbre))) then
                                --Si Parent est dans le sous-arbre gauche de la racine
                                    max:=NodeKey(Arbre)-1; --
la valeur maximale est celle de la racine -1
```

```
                                else  --Si Parent est dans le sous-arbre droit de la racine
                                    max:=NodeKey(Rech_Ancetre(NodeKey(Rech_Noeud(Parent,Arb
re)),Arbre))-1;
                                    --
La valeur maximale est celle du prédécesseur de Parent -1
                                end if;
                            end if;
                            --Intervalle du type [min,max]
                            Empiler(Ens1,min);
                            Empiler(Ens1,max);
                            return Ens1;
                        elsif not Est_Nul(Rech_Noeud(Parent,Arbre)) and then Fils_Droit(Rec
h_Ancetre(Parent,Arbre))=Rech_Noeud(Parent,Arbre) then
                            --Si le noeud Parent est le fils droit de son propre parent.
                            --Pour que la nouvelle clé soit le fils droit du noeud
                            minf:=False;M_inf:=False;
                            Temp:=Rech_Noeud(Parent,Arbre);
                            min:=NodeKey(Temp)+1;
                            while Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre and then Temp/=F
ils_Gauche(Rech_Ancetre(NodeKey(Temp),Arbre)) loop --
Tant que le prédécesseur de Temp est différent de la racine et que Temp n'est pas un fils g
auche
                                Temp:=Rech_Ancetre(NodeKey(Temp),Arbre);
                            end loop;
                            if Rech_Ancetre(NodeKey(Temp),Arbre)/=Arbre then --
Si le prédécesseur de Temp est différent de la racine
                                max:=NodeKey(Rech_Ancetre(NodeKey(Temp),Arbre))-1; --
La valeur maximale est celle du prédécesseur du premier prédécesseur étant un fils gauche -
1
                            else
                                if not Est_Nul(Rech_Noeud(Parent,Fils_Gauche(Arbre))) then
                                --Si Parent est dans le sous-arbre gauche de la racine
                                    max:=NodeKey(Arbre)-1; --
La valeur maximale est automatiquement la racine -1
                                else --Si Parent est dans le sous-arbre droit de la racine
                                    M_inf:=True; --
Les valeurs possibles ne sont pas majorées
                                end if;
                            end if;
                            Empiler(Ens1,min);
                            if M_inf then --Si les valeurs possibles ne sont pas majorées
                                Empiler(Ens1,0);Empiler(Ens1,-181199); --
Intervalle du type [min,+INFINI[
                            else --Si les valeurs possibles sont majorées
                                Empiler(Ens1,max); --Intervalle du type [min,max]
                            end if;
                            return Ens1;
                        else
```

```
                        return Vide;
                    end if;
                end if;
            else
                return Vide;
            end if;
        else
            return Vide;
        end if;
    end NewKeyInterval;

    procedure Inserer(Cle: in Integer; Donnee: in T_Value; Arbre: in out T_Branch) is
    begin
        if Arbre=Null then
            Arbre:= New T_Node'(Cle, Donnee, Null, Null);
        elsif not Est_Nul(Rech_Noeud(Cle,Arbre)) then
            Put_Line("Existe déjà!");
        elsif Cle=-181199 or Cle=-34404 then
            Put_Line("Veuillez saisir une autre clé, les clés -34404 et -
181199 sont utilisées intérieurement par ce programme pour assurer son fonctionnement..");
        else
            if Cle>Arbre.all.Cle then
                Inserer(Cle, Donnee, Arbre.all.FilsD);
            elsif Cle<Arbre.all.Cle then
                Inserer(Cle, Donnee, Arbre.all.FilsG);
            elsif Cle=Arbre.all.Cle then
                Arbre.all.Donnee:=Donnee;
            else
                Null;
            end if;
        end if;
    end Inserer;

    procedure Ajouter2(Cle_Nouveau_Noeud: in Integer; Donnee_Nouveau_Noeud: in T_Value; Cle
_Noeud_Parent:in integer; Arbre: in out T_Branch) is
        Noeud: T_Branch;
        Ens:T_Pile;
        NewKey:Integer:=Cle_Nouveau_Noeud;
        AjoutPossible: Boolean:=False;
        choix: Integer;
        NewParentKey,Grand_Ancestor:Integer;
        leftright:Character;
        begin
            if Arbre=Null then
                Arbre:=New T_Node;
                Arbre.Cle:=Cle_Noeud_Parent;
                Inserer(Cle_Nouveau_Noeud,Donnee_Nouveau_Noeud,Arbre);
            elsif Cle_Nouveau_Noeud=-181199 or Cle_Nouveau_Noeud=-34404 then
```

```
                Put_Line("Veuillez saisir une autre clé, les clés -34404 et -
181199 sont utilisées intérieurement par ce programme pour assurer son fonctionnement..");
            elsif Est_Nul(Rech_Noeud(Cle_Noeud_Parent,Arbre)) then --
Si le Parent n'existe même pas
                Put_Line("Il faut d'abord créer le prédécesseur!");
            elsif not Est_Nul(Rech_Noeud(Cle_Nouveau_Noeud,Arbre)) then --
Si l'arbre n'est pas vide et que le prédécesseur existe mais que la nouvelle clé existe déj
à
                Put_Line("Existe déjà!");
            elsif (not Est_Nul(Fils_Droit(Rech_Noeud(Cle_Noeud_Parent,Arbre)))) and (not Es
t_Nul(Fils_Gauche(Rech_Noeud(Cle_Noeud_Parent,Arbre)))) then --
Si le Parent a déjà deux fils
                Put_Line("Plus de place!");
            elsif Cle_Nouveau_Noeud>Cle_Noeud_Parent and not Est_Nul(Fils_Droit(Rech_Noeud(
Cle_Noeud_Parent,Arbre))) then
            --
Si la clé est supérieure à celle du parent et que ce dernier a déjà un fils droit
                if Est_Nul(Fils_Gauche(Rech_Noeud(Cle_Noeud_Parent,Arbre))) then --
Si le fils gauche est vide
                    Put_Line("Emplacement rempli! Ressayez avec une clé inférieure à celle
du prédécesseur.");
                else --Si le fils gauche n'est pas vide
                    Put_Line("Plus de place! Tentez plutôt une modification...");
                end if;
            elsif Cle_Nouveau_Noeud<Cle_Noeud_Parent and not Est_Nul(Fils_Gauche(Rech_Noeud
(Cle_Noeud_Parent,Arbre))) then
            --
Si la clé est inférieure à celle du parent et que ce dernier a déjà un fils gauche
                if Est_Nul(Fils_Droit(Rech_Noeud(Cle_Noeud_Parent,Arbre))) then --
SI le fils droit est vide
                    Put_Line("Emplacement rempli! Ressayez avec une clé supérieure à celle
du prédécesseur.");
                else
            --Si le fils droit n'est pas vide
                    Put_Line("Plus de place! Tentez plutôt une modification...");
                end if;
            else
                 Ens:=NewKeyInterval(Cle_Nouveau_Noeud,Cle_Noeud_Parent,Arbre); --
Pile contenant les bornes des valeurs possibles
                 if Piles_Cle.Sommet(Ens)=-
181199 and Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=0 and not Piles_Cle.Est_Vide(Piles_Cl
e.Next_Pile(Piles_Cle.Next_Pile(Ens))) then --Intervalle du type [min, +INFINI[
                    while NewKey < Piles_Cle.Sommet(Piles_Cle.Next_Pile(Piles_Cle.Next_Pile
(Ens))) or (NewKey=-181199 or NewKey=-34404 ) loop
                        Put("Valeur invalide! Doit être supérieur à " & Integer'Image(Piles
_Cle.Sommet(Piles_Cle.Next_Pile(Piles_Cle.Next_Pile(Ens)))) & ". Saisissez une autre valeur
 : ");
                        Get(NewKey);
```

```
                        New_Line;
                    end loop;
                    AjoutPossible:=True;
                elsif (Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=0 and (not Piles_Cle.Est_
Vide(Piles_Cle.Next_Pile(Piles_Cle.Next_Pile(Ens))))) and then Piles_Cle.Sommet(Piles_Cle.N
ext_Pile(Piles_Cle.Next_Pile(Ens)))=-181199 then --Intervalle du type ]-INFINI,max]
                    while Piles_Cle.Sommet(Ens)<NewKey  or (NewKey=-181199 or NewKey=-
34404 )loop
                        Put("Valeur invalide! Doit être inférieur à " & Integer'Image(Piles
_Cle.Sommet(Ens)) & ". Saisissez une autre valeur : ");
                        Get(NewKey);
                        New_Line;
                    end loop;
                    AjoutPossible:=True;
                elsif Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))<Piles_Cle.Sommet(Ens) THEN
 --Intervalle du type [min,max]
                    while NewKey < Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))  or Piles_Cle
.Sommet(Ens) < NewKey or (NewKey=-181199 or NewKey=-34404 ) loop
                        Put("Valeur invalide! Doit être entre " & Integer'Image(Piles_Cle.S
ommet(Piles_Cle.Next_Pile(Ens))) & " et" & Integer'Image(Piles_Cle.Sommet(Ens)) & ". Saisis
sez une autre valeur : ");
                        Get(NewKey);
                        New_Line;
                    end loop;
                    AjoutPossible:=True;
                elsif Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=Piles_Cle.Sommet(Ens) THEN
 --Une seule valeur possible
                    while NewKey /= Piles_Cle.Sommet(Ens) loop
                        Put("Valeur invalide! Doit être égale à " & Integer'Image(Piles_Cle
.Sommet(Ens)) & ". Saisissez cette valeur : ");
                        Get(NewKey);
                        New_Line;
                    end loop;
                    AjoutPossible:=True;
                else --Aucune valeur possible
                    Put_Line("Insertion impossible! Il va falloir modifier la clé: " & Inte
ger'Image(Cle_Noeud_Parent) & " (clé prédécesseur) ou multiplier toutes les clés de l'arbre
 par 10.");
                    Put_Line("1. Modifier la clé" & Integer'Image(Cle_Noeud_Parent));
                    Put_Line("2. Multiplier toutes les clés de l'arbre par 10");
                    Put("Choisissez une option: "); Get(choix); New_Line;
                    case choix is
                        when 1=>
                            Put("Saisissez la valeur de la nouvelle clé que vous voulez att
ribuer à " & Integer'Image(Cle_Noeud_Parent) & " : "); Get(NewParentKey); New_Line;
                            leftright:=Gauche_ou_Droite(Cle_Noeud_Parent,Arbre);
                            if leftright/='R' then
```

```
                                Grand_Ancestor:=Nodekey(Rech_Ancetre(Cle_Noeud_Parent,Arbre
));
                            end if;
                            while NewParentKey=-181199 or NewParentKey=-34404 loop
                                Put("Veuillez saisir une autre clé, les clés -34404 et -
181199 sont utilisées intérieurement par ce programme pour assurer son fonctionnement : ");
Get(NewParentKey);
                                New_Line;
                            end loop;
                            Modifier_Cle(Cle_Noeud_Parent, NewParentKey, Arbre);
                            Put_Line("Vous allez maintenant ressayer d'ajouter la première
clé (" & Integer'Image(NewKey) & " ).");
                            if leftright='G' then
                                NewParentKey:=Nodekey(Fils_Gauche(Rech_Noeud(Grand_Ancestor
,Arbre)));
                            elsif leftright='D' then
                                NewParentKey:=Nodekey(Fils_Droit(Rech_Noeud(Grand_Ancestor,
Arbre)));
                            else
                                NewParentKey:=Nodekey(Arbre);
                            end if;
                            Ajouter2(NewKey,Donnee_Nouveau_Noeud,NewParentKey,Arbre);
                        when 2=>
                            Multiplier_10(Arbre);
                            if Cle_Nouveau_Noeud>Cle_Noeud_Parent then --
Si on voulait que ça soit un fils droit
                                Put_Line("La clé" & Integer'Image(NewKey) & " que vous voul
iez ajouter à la clé" & Integer'Image(Cle_Noeud_Parent*10) & " est maintenant devenue" & In
teger'Image(Cle_Noeud_Parent*10 +5) & ".");
                                Put_Line("Essai d'ajouter" & Integer'Image(Cle_Noeud_Parent
*10 +5) & " à la clé" & Integer'Image(Cle_Noeud_Parent*10) & " :");
                                Ajouter2(Cle_Noeud_Parent*10 +5,Donnee_Nouveau_Noeud,Cle_No
eud_Parent*10,Arbre);
                            else --si on voulait que ça soit un fils gauche
                                Put_Line("La clé" & Integer'Image(NewKey) & " que vous voul
iez ajouter à la clé" & Integer'Image(Cle_Noeud_Parent*10) & " est maintenant devenue" & In
teger'Image(Cle_Noeud_Parent*10 -5) & ".");
                                Put_Line("Essai d'ajouter" & Integer'Image(Cle_Noeud_Parent
*10 -5) & " à la clé" & Integer'Image(Cle_Noeud_Parent*10) & " :");
                                Ajouter2(Cle_Noeud_Parent*10 -
5,Donnee_Nouveau_Noeud,Cle_Noeud_Parent*10,Arbre);
                            end if;
                        when others=> Put_Line("Option saisie invalide!");
                    end case;
                end if;
                if AjoutPossible then
                    Noeud:=Rech_Noeud(Cle_Noeud_Parent,Arbre);
                    Inserer(NewKey,Donnee_Nouveau_Noeud,Noeud);
```

```
                else
                    Null;
                end if;
            end if;
        end Ajouter2;


                        --MODIFICATION--

    procedure Modifier_Cle_Racine(NewCle:in Integer;Arbre: in out T_Branch) is
        begin
            if Arbre/=Null then
                if NewCle/=-181199 and NewCle/=-34404 then
                    Arbre.all.Cle:=NewCle;
                end if;
            end if;
        end Modifier_Cle_Racine;

    procedure Modifier_Cle(Cle,NewCle: in Integer; Arbre: in out T_Branch) is
        Noeud:T_Branch;
        Ens: T_Pile;
        FGKey,FDKey,TMPKEY,CleAncetre,NewParentKey,GAUCHEOUDROIT:Integer;
        NewKey:Integer:=NewCle;
        AjoutPossible:Boolean:=False;
        choix: Character:='n';
        begin
            if Arbre/=Null then
                if Cle=NewCle then
                    Null;
                elsif NewCle=-181199 or NewCle=-34404 then
                    Put_Line("Veuillez saisir une autre clé, les clés -34404 et -
181199 sont utilisées intérieurement par ce programme pour assurer son fonctionnement..");
                elsif not Est_Nul(Rech_Noeud(NewCle,Arbre)) then --Si NewCle existe déjà
                    Put_Line("Existe déjà!");
                else
                    if (not Est_Nul(Rech_Noeud(Cle,Arbre))) then --Si la clé existe d'abord
                        if not Est_Nul(Rech_Ancetre(Cle,Arbre)) then --
Si la clé a un prédécesseur
                            CleAncetre:=Rech_Ancetre(Cle,Arbre).all.Cle; --
Récupérer la clé du prédécesseur
                            if Nodekey(Fils_Gauche(Rech_Noeud(CleAncetre,Arbre)))=NewCle th
en --Si la clé qu'on veut modifer est un fils gauche
                                GAUCHEOUDROIT:=CleAncetre-1; --
Ceci indique à la fonction NewKeyInterval qu'on veut modifier un fils gauche
                            else --Si c'est un fils droit
                                GAUCHEOUDROIT:=CleAncetre+1; --
Ceci indique à la fonction NewKeyInterval qu'on veut modifier un fils droit
                            end if;
```

```
                            Ens:=NewKeyInterval(GAUCHEOUDROIT,CleAncetre,Arbre); --
Intervalle des valeurs possibles
                            if not Est_Nul(Fils_Droit(Rech_Noeud(Cle,Arbre))) then --
Si la clé possède un fils droit
                                FDKey:=Nodekey(Fils_Droit(Rech_Noeud(Cle,Arbre)))-1; --
clé du fils droit -1
                            end if;
                            if not Est_Nul(Fils_Gauche(Rech_Noeud(Cle,Arbre))) then --
Si la clé possède un fils gauche
                                FGKey:=Nodekey(Fils_Gauche(Rech_Noeud(Cle,Arbre)))+1; --
clé du fils gauche +1
                            end if;
                            if Piles_Cle.Sommet(Ens)=-
181199 and Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=0 and (not Piles_Cle.Est_Vide(Piles_C
le.Next_Pile(Piles_Cle.Next_Pile(Ens)))) and (not Est_Nul(Fils_Droit(Rech_Noeud(Cle,Arbre))
)) then -- intervalle du type [min,+inf[
                                Piles_Cle.Depiler(Ens);Piles_Cle.Depiler(Ens); --
intervalle devenu sans borne max
                                if (not Est_Nul(Fils_Gauche(Rech_Noeud(Cle,Arbre)))) and FG
Key>Piles_Cle.Sommet(Ens) then
                                -
- si la clé possède un fils gauche et que la clé de ce dernier rétraissit l'intervalle des
valeurs possibles
                                    Piles_Cle.Depiler(Ens);Piles_Cle.Empiler(Ens,FGKey);Pil
es_Cle.Empiler(Ens,FDKey);
                                    --intervalle devenu [FGkey,FDKey]
                                else
                                    Piles_Cle.Empiler(Ens,FDKey); --
intervalle devenu [min,FDKey]
                                end if;
                            elsif ((not Est_Nul(Fils_Gauche(Rech_Noeud(Cle,Arbre)))) and (P
iles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=0 and (not Piles_Cle.Est_Vide(Piles_Cle.Next_Pile
(Piles_Cle.Next_Pile(Ens)))))) and then Piles_Cle.Sommet(Piles_Cle.Next_Pile(Piles_Cle.Next
_Pile(Ens)))=-181199  then --intervalle du type ]-inf,max]
                                if (not Est_Nul(Fils_Droit(Rech_Noeud(Cle,Arbre)))) and FDK
ey<Piles_Cle.Sommet(Ens)then
                                --
Si la clé possède un fils droit et que la clé de ce dernier rétraissit l'intervalle des val
eurs possibles
                                    Piles_Cle.Depiler(Ens);Piles_Cle.Depiler(Ens);Piles_Cle
.Depiler(Ens);Piles_Cle.Empiler(Ens,FGKey);Piles_Cle.Empiler(Ens,FDKey);
                                    --intervalle devenu [FGKey,FDKey]
                                else
                                    TMPKEY:=Piles_Cle.Sommet(Ens); --préserver la borne max
                                     Piles_Cle.Depiler(Ens);Piles_Cle.Depiler(Ens);Piles_Cl
e.Depiler(Ens);Piles_Cle.Empiler(Ens,FGKey);Piles_Cle.Empiler(Ens,TMPKEY);
                                     --intervalle deveny [FGKey,max]
                                end if;
```

```
                            elsif Piles_Cle.Est_Vide(Piles_Cle.Next_Pile(Piles_Cle.Next_Pil
e(Ens))) and Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))<Piles_Cle.Sommet(Ens) and ((not Est
_Nul(Fils_Droit(Rech_Noeud(Cle,Arbre)))) or (not Est_Nul(Fils_Gauche(Rech_Noeud(Cle,Arbre))
))) THEN --intervalle du type [min,max] avec min<max
                                if (not Est_Nul(Fils_Droit(Rech_Noeud(Cle,Arbre)))) and FDK
ey<Piles_Cle.Sommet(Ens) then
                                    -
- si le fils droit existe et que sa clé rétraissit l'intervalle
                                    Piles_Cle.Depiler(Ens);Piles_Cle.Empiler(Ens,FDKey);
                                end if;
                                if (not Est_Nul(Fils_Gauche(Rech_Noeud(Cle,Arbre)))) and FG
Key>Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens)) then
                                    -
- si le fils gauche existe et que sa clé rétraissit l'intervalle
                                    TMPKEY:=Piles_Cle.Sommet(Ens);Piles_Cle.Depiler(Ens);Pi
les_Cle.Depiler(Ens);
                                    Piles_Cle.Empiler(Ens,FGKey);Piles_Cle.Empiler(Ens,TMPK
EY);
                                end if;
                            end if;
                            if Piles_Cle.Sommet(Ens)=-
181199 and Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=0 and not Piles_Cle.Est_Vide(Piles_Cl
e.Next_Pile(Piles_Cle.Next_Pile(Ens))) then -
- intervalle du type [min,+inf[
                                while NewKey < Piles_Cle.Sommet(Piles_Cle.Next_Pile(Piles_C
le.Next_Pile(Ens))) or (NewKey=-181199 or NewKey=-34404 )loop
                                    Put("Valeur invalide! Doit être supérieur à " & Integer
'Image(Piles_Cle.Sommet(Piles_Cle.Next_Pile(Piles_Cle.Next_Pile(Ens)))) & ". Saisissez une
autre valeur : ");
                                    Get(NewKey);
                                    New_Line;
                                end loop;
                                AjoutPossible:=True;
                            elsif (Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=0 and (not Pi
les_Cle.Est_Vide(Piles_Cle.Next_Pile(Piles_Cle.Next_Pile(Ens))))) and then Piles_Cle.Sommet
(Piles_Cle.Next_Pile(Piles_Cle.Next_Pile(Ens)))=-181199 then --intervalle du type ]-
inf,max]
                                while Piles_Cle.Sommet(Ens)<NewKey or (NewKey=-
181199 or NewKey=-34404 ) loop
                                    Put("Valeur invalide! Doit être inférieur à " & Integer
'Image(Piles_Cle.Sommet(Ens)) & ". Saisissez une autre valeur : ");
                                    Get(NewKey);
                                    New_Line;
                                end loop;
                                AjoutPossible:=True;
                            elsif Piles_Cle.Est_Vide(Piles_Cle.Next_Pile(Piles_Cle.Next_Pil
e(Ens))) and Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))<Piles_Cle.Sommet(Ens) THEN --
intervalle du type [min,max] avec min<max
```

```
                                while (NewKey < Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))
 or Piles_Cle.Sommet(Ens) < NewKey) or (NewKey=-181199 or NewKey=-34404 ) loop
                                    Put("Valeur invalide! Doit être entre " & Integer'Image
(Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))) & " et" & Integer'Image(Piles_Cle.Sommet(Ens))
 & ". Saisissez une autre valeur : ");
                                    Get(NewKey);
                                    New_Line;
                                end loop;
                                AjoutPossible:=True;
                            elsif Piles_Cle.Est_Vide(Piles_Cle.Next_Pile(Piles_Cle.Next_Pil
e(Ens))) and  Piles_Cle.Sommet(Piles_Cle.Next_Pile(Ens))=Piles_Cle.Sommet(Ens) THEN --
une seule valeur possible!
                                while NewKey /= Piles_Cle.Sommet(Ens) loop
                                    Put("Valeur invalide! Doit être égale à " & Integer'Ima
ge(Piles_Cle.Sommet(Ens)) & ". Saisissez cette valeur : ");
                                    Get(NewKey);
                                    New_Line;
                                end loop;
                                AjoutPossible:=True;
                            else --aucune valeur possible!
                                Put_Line("Modification impossible! Il va falloir modifier l
a clé: " & Integer'Image(CleAncetre) & " (clé prédécesseur).");
                                Put("Voulez-
vous la modifier? [y/n] :"); Get(choix); New_Line;
                                if choix='n' or choix='N' then
                                    Null;
                                else
                                    Put("Saisissez la valeur de la nouvelle clé que vous vo
ulez attribuer à " & Integer'Image(CleAncetre) & " : "); Get(NewParentKey); New_Line;
                                    while NewParentKey=-181199 or NewParentKey=-34404 loop
                                        Put("Veuillez saisir une autre clé, les clés -
34404 et -
181199 sont utilisées intérieurement par ce programme pour assurer son fonctionnement : ");
Get(NewParentKey);
                                        New_Line;
                                    end loop;
                                    Modifier_Cle(CleAncetre, NewParentKey, Arbre);
                                    Put_Line("Vous avez modifié la clé du prédécesseur.");
                                    Put_Line("Vous allez maintenant ressayer de modifier la
 première clé (" & Integer'Image(NewCle) & " ).");
                                    Modifier_Cle(Cle,NewCle,Arbre);
                                    AjoutPossible:=False;
                                end if;
                            end if;
                            if AjoutPossible then
                                Noeud:=Rech_Noeud(Cle,Arbre);
                                Noeud.all.Cle:=NewKey;
                            else
```

```
                                Null;
                            end if;
                        else --
si la clé existe mais n'a pas d'ancêtre (i.e., c'est la racine)
                            Arbre.all.Cle:=NewKey;
                        end if;
                    else --Si la clé n'existe pas.
                        Put_Line("Inexistante!");
                    end if;
                end if;
            else -- si l'arbre est vide.
                Initialiser(NewCle,Arbre);
            end if;
        end Modifier_Cle;

    procedure Modifier_Donnee(Cle: in Integer; NewDonnee: in T_Value; Arbre: in out T_Branc
h) is
        Noeud:T_Branch;
        begin
            if Arbre/= Null then
                if not Est_Nul(Rech_Noeud(Cle,Arbre)) then
                    Noeud:=Rech_Noeud(Cle,Arbre);
                    Noeud.all.Donnee:=NewDonnee;
                else
                    Inserer(Cle,NewDonnee,Arbre);
                end if;
            else
                Arbre:= new T_Node'(Cle, NewDonnee,Null,Null);
            end if;
        end Modifier_Donnee;



                        --AFFICHAGE--

    procedure Afficher_ABR(Arbre: in T_Branch) is
        Original: constant T_Branch:=Arbre;
        procedure Afficher_Fils(Arbre: in T_Branch) is
            begin
                if Arbre=Null then
                    Null;
                else
                    if Arbre.all.FilsG/=Null then
                        for i in 1..Gen(Arbre.all.FilsG.all.Cle,Original) loop
                            Put("    ");
                        end loop; --Ajoutera des tabulations tant que le noeud est profond
                        Put("  --");
                        Afficher_Donnee(Arbre.all.FilsG.all.Donnee);
                        Put(" :");
```

```
                    Put(Integer'Image(Arbre.all.FilsG.all.Cle));
                    New_Line;
                    Afficher_Fils(Arbre.all.FilsG);
                else
                    Null;
                end if;
                if Arbre.all.FilsD/=Null then
                    for i in 1..Gen(Arbre.all.FilsD.all.Cle,Original) loop
                        Put("    ");
                    end loop;
                    Put("  --");
                    Afficher_Donnee(Arbre.all.FilsD.all.Donnee);
                    Put(" :");
                    Put(Integer'Image(Arbre.all.FilsD.all.Cle));
                    New_Line;
                    Afficher_Fils(Arbre.all.FilsD);
                else
                    Null;
                end if;
            end if;
        end Afficher_Fils;

    begin
        if Arbre=Null then
            New_Line;
        else
            Put_Line(Integer'Image(Arbre.all.Cle)); --Afichage de la racine
            Afficher_Fils(Arbre); --Affichage du reste de l'arbre

        end if;
    end Afficher_ABR;

    procedure Afficher_APartir(Cle:in Integer; Arbre: in T_Branch) is
        Original: constant T_Branch:=Rech_Noeud(Cle,Arbre); --
Noeud contenant la clé Cle
        procedure Afficher_Fils(Arbre: in T_Branch) is
            begin
                if Arbre=Null then
                    Null;
                else
                    if Arbre.all.FilsG/=Null then
                        for i in 1..Gen(Arbre.all.FilsG.all.Cle,Original) loop
                            Put("    ");
                        end loop;
                        Put("--");
                        Afficher_Donnee(Arbre.all.FilsG.all.Donnee);
                        Put(" :");
                        Put(Integer'Image(Arbre.all.FilsG.all.Cle));
```

```
                        New_Line;
                        Afficher_Fils(Arbre.all.FilsG);
                    else
                        Null;
                    end if;
                    if Arbre.all.FilsD/=Null then
                        for i in 1..Gen(Arbre.all.FilsD.all.Cle,Original) loop
                            Put("    ");
                        end loop;
                        Put("--");
                        Afficher_Donnee(Arbre.all.FilsD.all.Donnee);
                        Put(" :");
                        Put(Integer'Image(Arbre.all.FilsD.all.Cle));
                        New_Line;
                        Afficher_Fils(Arbre.all.FilsD);
                    else
                        Null;
                    end if;
                end if;
            end Afficher_Fils;

        begin
            if Arbre=Null then
                New_Line;
            else
                Put_Line(Integer'Image(Original.all.Cle));
                Afficher_Fils(Original);

            end if;
        end Afficher_APartir;

end Arbre_Binaire;
```