```
with arbre_binaire;


procedure test_arbre_binaire is

    package ABR_Char is
        new Arbre_Binaire (T_Value => Character, Zero => '0');
    use ABR_Char;

    A: T_Branch;

    procedure Exemple_Arbre(A:in out T_Branch) is
        begin
            Initialiser(20,A);
            Inserer(22,'A',A);
            Inserer(23,'B',A);
            Inserer(16,'C',A);
            Inserer(10,'D',A);
            Inserer(21,'E',A);
        end Exemple_Arbre;

    procedure Tester_Exemple_Arbre is
        begin
            Exemple_Arbre(A);
            --
Tester l'existence et la bonne répartition des clés au sein de l'arbre.
                --
En utilisant les fonctions NodeKey, NodeValue, Fils_Gauche, Fils_Droit et Est_
Nul;
            pragma assert(not Est_Nul(A));                                  --
S'assurer que A n'est pas vide.
            pragma assert(NodeKey(A)=20);                                   --
S'assurer que la clé de la racine est 20.
            pragma assert(not Est_Nul(Fils_Gauche(A)));                     --
S'assurer que le fils gauche n'est pas vide.
            pragma assert(NodeKey(Fils_Gauche(A))=16);                      --
S'assurer que sa clé est 16.
            pragma assert(NodeValue(Fils_Gauche(A))='C');                   --
S'assurer que sa donnée est 'C'.
            pragma assert(not Est_Nul(Fils_Gauche(Fils_Gauche(A))));        --
S'assurer que le fils gauche de 16 n'est pas vide.
            pragma assert(Est_Nul(Fils_Droit(Fils_Gauche(A))));             --
S'assurer que le fils droit de 16 est vide.
            pragma assert(NodeKey(Fils_Gauche(Fils_Gauche(A)))=10);         --
S'assurer que la clé du fils gauche de 16 est 10.
            pragma assert(NodeValue(Fils_Gauche(Fils_Gauche(A)))='D');      --
S'assurer que la donnée du fils gauche de 16 est 'D'.
            pragma assert(not Est_Nul(Fils_Droit(A)));                      --
S'assurer que le fils droit n'est pas vide.
```

```
            pragma assert(NodeKey(Fils_Droit(A))=22);                          --
S'assurer que sa clé est 22.
            pragma assert(NodeValue(Fils_Droit(A))='A');                       --
S'assurer que sa donnée est 'A'.
            pragma assert(not Est_Nul(Fils_Droit(Fils_Droit(A))));             --
S'assurer que son fils droit n'est pas vide.
            pragma assert(NodeKey(Fils_Droit(Fils_Droit(A)))=23);              --
S'assurer que la clé du fils droit de 22 est 23.
            pragma assert(NodeValue(Fils_Droit(Fils_Droit(A)))='B');           --
S'assurer que la donnée du fils droit de 22 est 'B'.
            pragma assert(not Est_Nul(Fils_Gauche(Fils_Droit(A))));            --
S'assurer que le fils gauche de 22 n'est pas vide.
            pragma assert(NodeKey(Fils_Gauche(Fils_Droit(A)))=21);             --
S'assurer que la clé du fils gauche de 22 est 21.
            pragma assert(NodeValue(Fils_Gauche(Fils_Droit(A)))='E');          --
S'assurer que la donnée du fils gauche de 22 est 'E'.

                --En utilisant la fonction Rech_Noeud.
            pragma assert(not Est_Nul(Rech_Noeud(21,A)));    --
S'assurer que la clé 21 existe dans l'arbre.
            pragma assert(not Est_Nul(Rech_Noeud(10,A)));    --
S'assurer que la clé 10 existe dans l'arbre.
            pragma assert(Est_Nul(Rech_Noeud(69,A)));        --
S'assurer que la clé 69 n'existe pas dans l'arbre.
                --En utilisant la fonction Rech_Ancetre.
            pragma assert(NodeKey(Rech_Ancetre(16,A))=20);   --
S'assurer que l'ancêtre de 16 est 20.
            pragma assert(NodeKey(Rech_Ancetre(10,A))=16);   --
S'assurer que l'ancêtre de 10 est 16.
            pragma assert(NodeKey(Rech_Ancetre(22,A))=20);   --
S'assurer que l'ancêtre de 22 est 20.
            pragma assert(NodeKey(Rech_Ancetre(21,A))=22);   --
S'assurer que l'ancêtre de 21 est 22.
            pragma assert(NodeKey(Rech_Ancetre(23,A))=22);   --
S'assurer que l'ancêtre de 23 est 22.

                --En utilisant la fonction Gen.
            pragma assert(Gen(21,A)=2);                      --
S'assurer que la génération de 21 est 2.
            pragma assert(Gen(23,A)=2);                      --
S'assurer que la génération de 23 est 2.
            pragma assert(Gen(10,A)=2);                      --
S'assurer que la génération de 10 est 2.
            pragma assert(Gen(16,A)=1);                      --
S'assurer que la génération de 16 est 1.
            pragma assert(Gen(22,A)=1);                      --
S'assurer que la génération de 22 est 1.
```

```
            pragma assert(Gen(20,A)=0);                         --
S'assurer que la génération de 2. est 0.

                --En utilisant la fonction Nbr_Fils_Noeud.
            pragma assert(Nbr_Fils_Noeud(20,A)=5);              --
S'assurer que le nombre de fils de 20 est 5.
            pragma assert(Nbr_Fils_Noeud(16,A)=1);              --
S'assurer que le nombre de fils de 16 est 1.
            pragma assert(Nbr_Fils_Noeud(10,A)=0);              --
S'assurer que le nombre de fils de 10 est 0.
            pragma assert(Nbr_Fils_Noeud(22,A)=2);              --
S'assurer que le nombre de fils de 22 est 2.
            pragma assert(Nbr_Fils_Noeud(21,A)=0);              --
S'assurer que le nombre de fils de 21 est 0.
            pragma assert(Nbr_Fils_Noeud(23,A)=0);              --
S'assurer que le nombre de fils de 23 est 0.

                --En utilisant Nbr_Meme_Generation.
            pragma assert(Nbr_Meme_Generation(0,A)=1);          --
S'assurer que le nombre de clés de génération 0 est 1.
            pragma assert(Nbr_Meme_Generation(1,A)=2);          --
S'assurer que le nombre de clés de génération 1 est 2.
            pragma assert(Nbr_Meme_Generation(2,A)=3);          --
S'assurer que le nombre de clés de génération 2 est 3.
        end Tester_Exemple_Arbre;

    procedure Tester_Supprimer is
        begin
            Detruire(A);
            pragma assert(Est_Nul(A));                          --
S'assurer que A a été détruit.
            Exemple_Arbre(A);
            Supprimer_Cle_ET_Fils(10,A);
            pragma assert(Est_Nul(Rech_Noeud(10,A)));           --
S'assurer que le noeud de clé 10 a été supprimé.
            Supprimer_Cle_ET_Fils(22,A);
            pragma assert(Est_Nul(Fils_Droit(A)));              --
S'assurer que le fils droit et ses fils ont été supprimés.
            Supprimer_Cle_ET_Fils(20,A);
            pragma assert(Est_Nul(A));                          --
S'assurer que A a été détruit.
        end Tester_Supprimer;

    begin
        Tester_Exemple_Arbre;
        Tester_Supprimer;
    end test_arbre_binaire;
```