I arbres binaires

On définit un arbre binaire par :

```
type arbin=^noeud;
     noeud=record
             filsg,filsd:arbin;
             cle:longint
           end;
```

Un arbre binaire `a` peut être vide (si `a=nil`). Sinon il est formé d'un noeud (`a^`) contenant une clé (`a^.cle`) et deux sous-arbres (`a^.filsg` et `a^.filsd`) qui peuvent chacun être vide ou non. Compléter les procédures et fonctions suivantes :

```
function nbnoeud(a:arbin):integer;
function hauteur(a:arbin):integer;
procedure stat(a:arbin;var nbnoeud:integer;var longmoy,longvar:real);
procedure affpref(a:arbin);
procedure affpost(a:arbin);
procedure affinf(a:arbin);
procedure affarbre(a:arbin);
function cree(fg:arbin;cle:longint;fd:arbin):arbin;
function copie(a:arbin):arbin;
function copiemiroir(a:arbin):arbin;
procedure libere(a:arbin);
```

`nbnoeud(a)` est le nombre de noeuds de l'arbre `a`. `hauteur(a)` est la hauteur de l'arbre `a`. Après l'appel `stat(a,n,m,v)`, la variable `n` contient le nombre de noeuds de l'arbre `a`. Si `n` est non nul (c'est-à-dire si `a` $\neq$ `nil`) alors `m` et `v` doivent contenir la moyenne et la variance de la distance de chacun des noeuds à la racine. Les appels `affpref(a)`, `affpost(a)` et `affinf(a)` affichent tous les trois l'ensemble des clés de tous les noeuds de l'arbre `a`, mais dans des ordres différents : `affpref` affiche la clé d'un noeud avant toutes les clés de tous les noeuds de ses deux fils. `affpost` affiche la clé d'un noeud après toutes les clés de tous les noeuds de ses deux fils. `affinf` affiche la clé d'un noeud après toutes les clés de tous les noeuds de son fils gauche, mais avant toutes les clés de tous les noeuds de son fils droit. La procédure `affarbre` affiche un arbre sous la forme suivante :

```
     /--| 19
     |  |  /--| 36
     |  \--| 37
  /--| 40
  |  |  /--| 45
  |  \--| 47
--| 62
  \--| 86
     \--| 96
```

`cree(a,5,b)` crée un nouveau noeud dont les champs sont initialisés avec les valeurs `a`, `b` et 5. Par exemple, l'instruction `a:=cree(cree(nil,12,nil),34,cree(nil,56,nil))` crée un arbre contenant trois noeuds. `copie(a)` est un arbre ayant exactement le même nombre de noeuds que `a`. Chacun de ses noeuds correspond à un noeud de `a` et a la

même clé que lui. Ces noeuds ont entre eux les mêmes relations de filiation que dans `a`. Les noeuds de `copie(a)` doivent être nouveaux (Ils peuvent être produit par `cree`). `copiemiroir` fait la même chose que `copie`, sauf que dans la copie, les fils gauches sont devenus des fils droits et réciproquement. `libere(a)` applique la procédure `dispose` à chacun des noeuds de l'arbre `a`. Toutes les procédures et fonctions précédentes sauf `cree` sont récursives et doivent avoir un temps de calcul en $O(n)$.

II arbres binaires de recherche

II.1) Un arbre binaire est un arbre binaire de recherche si les clés des noeuds de cet arbre, prises dans l'ordre infixe, sont dans l'ordre croissant. Compléter les procédures et fonctions suivantes :

```
function derecherche(a:arbin):boolean;
function presente(a:arbin;cle:longint):boolean;
function cherchecle(a:arbin;cle:longint):arbin;
procedure ajoutecle(var a:arbin;cle:longint);
procedure enlevecle(var a:arbin;cle:longint);
```

`derecherche(a)` indique si `a` est un arbre de recherche, c'est-à-dire si les clés de ses noeuds sont dans l'ordre croissant. `presente(a,x)` indique si `x` est la clé d'un des noeuds de l'arbre `a`. `cherchecle(a,x)` est un pointeur sur le noeud de l'arbre `a` dont la clé vaut `x`. `ajoutecle(a,x)` modifie l'arbre de recherche `a` en y ajoutant un noeud dont la clé vaut `x`. Ce noeud doit être ajouté comme fils d'un noeud qui n'avait pas encore tous ses fils, choisi de telle sorte que l'arbre reste un arbre de recherche. `enlevecle(a,x)` enlève le noeud de clé `x` de l'arbre `a`.

II.2) On définit deux suites récurrentes par $u_0 = 1$, $u_1 = 2$, $u_2 = 3$, $u_3 = 24$, $v_n = u_n * u_{n+1}$ et $u_{n+4} = (v_n + v_{n+2}) \bmod 2003 + 1$ pour n entier naturel. Ecrire un programme qui cherche le premier nombre qui apparaît deux fois dans la suite v. On pourra mettre les valeurs v_0, v_1, v_2 etc. dans un arbre de recherche.

Corrigé

```
(*$n+*)
type real=extended;
     arbin=^noeud;
     noeud=record
             filsg,filsd:arbin;
             cle:longint
           end;
function nbnoeud(a:arbin):integer;
begin
  if a=nil then nbnoeud:=0 else
  with a ^do
  nbnoeud:=1+nbnoeud(filsg)+nbnoeud(filsd)
end;
function hauteur(a:arbin):integer;
var hg,hd:integer;
begin
  if a=nil then hauteur:=0 else
  with a ^do
  begin
    hg:=hauteur(filsg);
    hd:=hauteur(filsd);
    if hg<=hd then hauteur:=hd+1
              else hauteur:=hg+1
  end
end;
procedure stat(a:arbin;var nbnoeud:integer;var longmoy,longvar:real);
var ng,nd:integer;
    mg,md,vg,vd:real;
begin
  if a=nil then
  begin
    nbnoeud:=0;
    longmoy:=0;
    longvar:=0
  end      else
  with a^do
  begin
    stat(filsg,ng,mg,vg);
    stat(filsd,nd,md,vd);
    mg:=mg+1;
    md:=md+1;
    nbnoeud:=ng+nd+1;
    longmoy:=(ng*mg+nd*md)/nbnoeud;
```

```
    longvar:=(ng*(sqr(mg)+vg)+nd*(sqr(md)+vd))/nbnoeud-sqr(longmoy)
  end
end;
procedure affpref(a:arbin);
begin
  if a<>nil then
  with a^ do
  begin
    write(cle,' ');
    affpref(filsg);
    affpref(filsd)
  end
end;
procedure affpost(a:arbin);
begin
  if a<>nil then
  with a^ do
  begin
    affpost(filsg);
    affpost(filsd);
    write(cle,' ')
  end
end;
procedure affinf(a:arbin);
begin
  if a<>nil then
  with a^ do
  begin
    affinf(filsg);
    write(cle,' ');
    affinf(filsd)
  end
end;
procedure affarbre(a:arbin);
type chaine=string;
procedure recurre(h,m,b:chaine;a:arbin);
begin
  if a<>nil then
  with a^do
  begin
    recurre(h+'   ',h+'/--',h+'|  ',filsg);
    writeln(m,'| ',cle);
    recurre(b+'|  ',b+'-',b+'   ',filsd)
  end
```

```
end;
begin
  recurre('  ','--','  ',a)
end;
function cree(fg,fd:arbin;cle:longint):arbin;
var a:arbin;
begin
  new(a);
  cree:=a;
  a^.filsg:=fg;
  a^.filsd:=fd;
  a^.cle:=cle
end;
function copie(a:arbin):arbin;
begin
  if a=nil then copie:=nil else
  with a^ do
  copie:=cree(copie(filsg),copie(filsd),cle)
end;
function copiemiroir(a:arbin):arbin;
begin
  if a=nil then copiemiroir:=nil else
  with a^ do
  copiemiroir:=cree(copiemiroir(filsg),copiemiroir(filsd),cle)
end;
procedure libere(a:arbin);
begin
  if a<>nil then
  with a^ do
  begin
    libere(filsg);
    libere(filsd);
    dispose(a)
  end
end;
function derecherche(a:arbin):boolean;
var premier:boolean;
    x:longint;
procedure recurre(a:arbin);
begin
  if a<>nil then
  with a^ do
  begin
    recurre(filsg);
```

```
    if premier then premier:=false else
    if cle<=x then derecherche:=false;
    x:=cle;
    recurre(filsd)
  end
end;
begin
  premier:=true;
  derecherche:=true;
  recurre(a)
end;
function presente(a:arbin;cle:longint):boolean;
begin
  if a=nil then presente:=false else
  if cle=a^.cle then presente:=true else
  if cle<a^.cle then presente:=presente(a^.filsg,cle)
                else presente:=presente(a^.filsd,cle)
end;
function cherchecle(a:arbin;cle:longint):arbin;
begin
  if a=nil then cherchecle:=nil else
  if cle=a^.cle then cherchecle:=a else
  if cle<a^.cle then cherchecle:=cherchecle(a^.filsg,cle)
                else cherchecle:=cherchecle(a^.filsd,cle)
end;
procedure ajoutecle(var a:arbin;cle:longint);
begin
  if a=nil then a:=cree(nil,nil,cle) else
  if cle=a^.cle then else
  if cle<a^.cle then ajoutecle(a^.filsg,cle)
                else ajoutecle(a^.filsd,cle)
end;
procedure enlevecle(var a:arbin;cle:longint);
function prendgauche(var a:arbin):arbin;
begin
  if a^.filsg<>nil then prendgauche:=prendgauche(a^.filsg) else
  begin
    prendgauche:=a;
    a:=a^.filsd
  end
end;
var b:arbin;
begin
  if a<>nil then
```

```
  if cle<a^.cle then enlevecle(a^.filsg,cle) else
  if cle>a^.cle then enlevecle(a^.filsd,cle) else
  if a^.filsg=nil then
  begin
    b:=a^.filsd;
    dispose(a);
    a:=b
  end             else
  if a^.filsg=nil then
  begin
    b:=a^.filsg;
    dispose(a);
    a:=b
  end             else
  begin
    b:=prendgauche(a^.filsd);
    b^.filsd:=a^.filsd;
    b^.filsg:=a^.filsg;
    dispose(a);
    a:=b
  end
end;
var u:array[0..4] of longint;
    i,n:integer;
    a:arbin;
begin
  u[0]:=1;
  u[1]:=2;
  u[2]:=3;
  u[3]:=24;
  n:=0;
  while not presente(a,u[0]*u[1]) do
  begin
    ajoutecle(a,u[0]*u[1]);
    inc(n);
    u[4]:=(u[0]*u[1]+u[2]*u[3]) mod 2003+1;
    for i:=0 to 3 do u[i]:=u[i+1]
  end;
  writeln(n,' ',u[0]*u[1])
end.
```