1) Dans un programme contenant les déclarations :

```
type anneau=^elem;
     elem=record
            prec,suiv:anneau;
            cle:integer
          end;
```

completez les fonctions et procédures suivantes.

`function cree(cle:integer):anneau;`

Cette fonction crée un nouvel anneau ne contenant qu'un seul élément (alloué par `new`).

`procedure croise(a,b:anneau);`

Cette procédure coupe le lien entre `a^` et son successeur ainsi que celui entre `b^` et son successeur, puis recrée un premier lien entre `a^` et l'ancien successeur de `b^` et un second lien entre `b^` et l'ancien successeur de `a^`. Cette procédure permet donc de couper un anneau en deux, ou de fusionner deux anneaux en un seul. En particulier `croise(a,cree(10))` insère un nouveau nouvel élément ayant pour clé 10 dans l'anneau contenant l'élément `a^`, juste après celui-ci.

`procedure detache(a:anneau);`

Cette procédure enlève l'élément `a^` de l'anneau dans lequel il se trouvait, et en fait un anneau à un élément. En fait il coupe un anneau en deux, ce qui peut être fait par un appel à `croise`.

`procedure retourne(a:anneau);`

Cette procédure échange les champs `suiv` et `prec` de chacun des éléments de l'anneau contenant `a^`. Après cela, cet anneau a les mêmes éléments, mais dans l'ordre inverse.

`function memeanneau(a,b:anneau):boolean;`

Cette fonction a pour valeur vrai si et seulement si `a^` et `b^` sont des éléments du même anneau.

`function longueur(a:anneau):integer;`

Cette fonction rend le nombre d'éléments de l'anneau contenant `a^`.

`function dist(a,b:anneau):integer;`

Cette fonction rend -1 si `a^` et `b^` appartiennent à des anneaux différents. Elle rend 0 si `a=b`. Si `a^` et `b^` sont deux éléments différents du même anneau, alors le résultat de la fonction est le nombre minimum de `^.suiv` à faire à partir de `a` pour atteindre `b`.

`function dist2(a,b:anneau):integer;`

Cette fonction rend -1 si `a^` et `b^` appartiennent à des anneaux différents. Elle rend 0 si `a=b`. Si `a^` et `b^` sont deux éléments différents du même anneau, alors le résultat de la fonction est le nombre minimum de `^.suiv` ou de `^.prec` à faire à partir de `a` pour atteindre `b`.

`procedure echange(a,b:anneau);`

Cette procédure échange les places des éléments `a^` et `b^` dans leurs anneaux respectifs. Elle doit marcher même si `a^` et `b^` sont seuls dans leur anneau, ou si ils sont consécutifs dans le même anneau. Elle peut se réaliser par deux appels à `croise`.

2) Qu'écrit le programme suivant :

```
type anneau=^elem;
     elem=record ...
function cree(cle:integer):anneau;
...
procedure croise(a,b:anneau);
...
function dist2(a,b:anneau):integer;
...
procedure echange(a,b:anneau);
...
procedure affiche(a:anneau);
var b:anneau;
begin
  b:=a;
  repeat
    write(a^.cle,' ');
    a:=a^.suiv
  until a=b;
  writeln
end;
var a,b:anneau;
    i:integer;
begin
  a:=cree(1);
  affiche(a);
  for i:=2 to 10 do croise(a,cree(i));
  affiche(a);
  retourne(a);
  affiche(a);
  b:=a^.suiv^.suiv^.suiv;
  affiche(b);
  writeln(dist(a,b),' ',dist(b,a),' ',dist2(a,b),' ',dist2(b,a));
  echange(a,b);
  affiche(a);
  affiche(b);
  croise(a,b);
  writeln(dist(a,b),' ',dist(b,a),' ',dist2(a,b),' ',dist2(b,a));
  affiche(a);
  affiche(b)
end.
```

Corrigé

```
type anneau=^elem;
     elem=record
            prec,suiv:anneau;
            cle:integer
          end;
function cree(cle:integer):anneau;
var a:anneau;
begin
  new(a);
  a^.cle:=cle;
  a^.suiv:=a;
  a^.prec:=a;
  cree:=a
end;
procedure croise(a,b:anneau);  (* swap(a^.suiv,b^.suiv) *)
var c,d:anneau;
begin
  c:=a^.suiv;
  d:=b^.suiv;
  c^.prec:=b;
  d^.prec:=a;
  a^.suiv:=d;
  b^.suiv:=c
end;
procedure detache(a:anneau);
begin
  croise(a^.prec,a)
end;
procedure retourne(a:anneau);
var b,c:anneau;
begin
  b:=a;
  repeat
    c:=a^.prec;
    a^.prec:=a^.suiv;
    a^.suiv:=c;
    a:=c
  until a=b
end;
function memeanneau(a,b:anneau):boolean;
var c:anneau;
begin
  c:=a;
```

```
  repeat
    a:=a^.suiv
  until (a=c) or (a=b);
  memeanneau:=a=b
end;
function longueur(a:anneau):integer;
var b:anneau;
    n:integer;
begin
  b:=a;
  n:=0;
  repeat
    n:=n+1;
    a:=a^.suiv
  until a=b;
  longueur:=n
end;
function dist(a,b:anneau):integer;
(* -1  si A et B ne sont pas sur le meme anneau
    0  si A=B
    sinon le nombre minimum de  ^.suiv a faire a partir de A pour
          arriver a B *)
var c:anneau;
    n:integer;
begin
  n:=0;
  c:=a;
  if a<>b then
  repeat
    n:=n+1;
    a:=a^.suiv
  until (a=b) or (a=c);
  if a=b then dist:=n
         else dist:=-1
end;
function dist2(a,b:anneau):integer;
(* -1  si A et B ne sont pas sur le meme anneau
    0  si A=B
    sinon le nombre minimum de ^.suiv ou de ^.prec a faire
          a partir de A pour arriver a B *)
var c,d:anneau;
    n:integer;
begin
  n:=0;
```

```
  c:=a;
  d:=b;
  if a<>b then
  repeat
    n:=n+1;
    c:=c^.suiv;
    d:=d^.suiv
  until (c=a) or (c=b) or (d=a) or (d=b);
  if (c=b) or (d=a) then dist2:=n
                    else dist2:=-1
end;
procedure echange(a,b:anneau);
(* echange les places des elements A et B *)
begin
  croise(a,b);
  croise(a^.prec,b^.prec)
end;
procedure aff(a:anneau);
var b:anneau;
begin
  b:=a;
  repeat
    write(a^.cle,' ');
    a:=a^.suiv
  until a=b;
  writeln
end;
procedure test(a:anneau);
var b:anneau;
begin
  b:=a;
  repeat
    if a^.suiv^.prec<>a then
    begin
      writeln(a^.cle,'^.suiv^.prec pas bon');
      repeat
        readln
      until false
    end;
    a:=a^.suiv
  until a=b
end;
procedure affab(a,b:anneau;noma,nomb:char);
begin
```

```
  test(a);
  test(b);
  if memeanneau(a,b) then write(noma,'=',nomb)
                     else write(noma,'#',nomb);
  write(dist(a,b):5,dist(b,a):3,dist2(a,b):3,'   |',noma,'|=',
       longueur(a):2,'   ');
  aff(a)
end;
var i:integer;
    a,b,c:anneau;
procedure affabc;
begin
  affab(a,b,'a','b');
  affab(b,c,'b','c');
  affab(c,a,'c','a');
  readln
end;
begin
  a:=cree(1);
  for i:=2 to 10 do croise(a,cree(i));
  b:=cree(21);
  for i:=22 to 24 do croise(b,cree(i));
  c:=a^.suiv;
  affabc;
  retourne(a);
  affabc;
  echange(a,c);
  affabc;
  echange(a,b);
  affabc;
  detache(c);
  affabc
end.
```