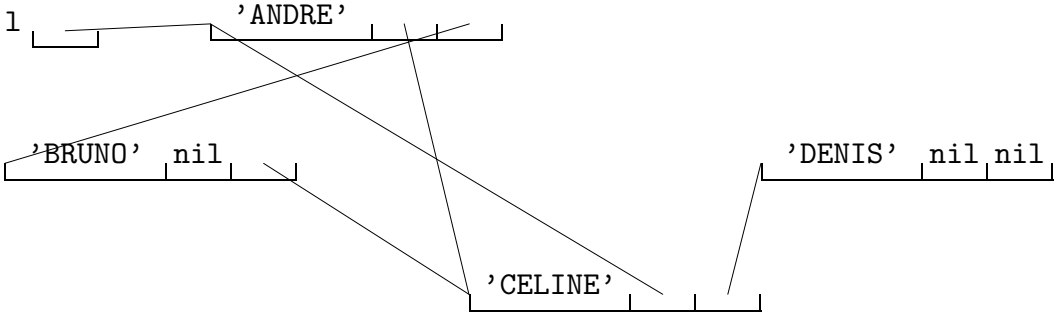
1) Dans un programme contenant les déclarations :

```
type pftd=^ftd;
     ftd=record
           nom,prenom:string[30];
           notes:array[1..5] of integer;
           my:integer;
           binome,suivant:pftd
         end;
var l:pftd;
```

on veut s'allouer dans le tas la place pour quatre fiches de T.D., mettre à jour les champs `nom`, `binome` et `suivant` de chacune de ces quatre fiches et mettre dans `l` un pointeur sur la première pour obtenir :



André et Céline forment un binôme alors que Bruno et Denis sont individualistes. On peut réaliser cela par les instructions suivantes :

```
new(l);
new(l^.suivant);
new(l^.suivant^.suivant);
l^.nom:='ANDRE';
l^.binome:=l^.suivant^.suivant;
l^.suivant^.binome:=nil;
```

Compléter ces instructions. (Il manque un `new` et six affectations)

2) On peut aussi procéder en utilisant des variables de type pointeur, ce qui rend le programme plus lisible :

```
var andre,bruno,celine,denis:pftd;
begin
  new(andre);
  new(bruno);
  new(celine);
  new(denis);
  l:=andre;
  andre^.nom:='ANDRE';
  andre^.binome:=celine;
  andre^.suivant:=bruno;
```

Compléter ces instructions. (Il manque neuf affectations)

3) On utilise la procédure récursive suivante pour afficher la liste des noms.

```
procedure affnom(a:pftd);
begin
  if a<>nil then
  begin
    writeln(a^.nom);
    affnom(a^.suivant)
  end
end;
```

Qu'apparaît-il sur l'écran quand on appelle la procédure par `affnom(l)` ? Ecrire une version non récursive de `affnom`. Que se passe-t-il si on intervertit les deux instructions `writeln(a^.nom)` et `affnom(a^.suivant)` ? Peut-on alors écrire facilement une version non récursive de cette procédure ?

4) Compléter la fonction
`function nbfiches(a:pftd):integer;`
qui donne le nombre de fiches dans la liste `a`. Par exemple `nbfiches(l)=4` tandis que `nbfiches(l^.suivant)=3`. Donner une version récursive et une version non récursive de cette fonction.

5) Compléter la fonction
`function nbbinomes(a:pftd):integer;`
qui donne le nombre de binômes dans la liste `a`. Par exemple `nbbinomes(l)=1`. On pourra compter les étudiants qui sont dans un binôme, puis diviser ce nombre par deux.

6) Compléter la procédure
`procedure affbinomes(a:pftd);`
qui affiche les binômes existants. Par exemple `affbinomes(l)` doit afficher un des deux couples `ANDRE+CELINE` ou `CELINE+ANDRE` mais pas les deux.

7) Compléter la fonction
`function copie(a:pftd):pftd;`
qui fait une copie de la liste de fiches de TD `a`. Cette copie doit être constituée de nouvelles fiches de TD (obtenues par `new`). Dans la copie tous les champs `binome` seront mis à `nil`, c'est à dire que les binômes seront séparés. Il est plus simple d'écrire cette fonction récursivement.

8) Ecrire maintenant la fonction `copie` itérativement et en lui faisant mettre à jour les champs `binome` de manière à conserver le même appariement en binômes que dans la liste originale. Il faut noter que les champs `binome` de la copie doivent pointer dans la copie et non dans l'original. On pourra écrire une fonction qui parcourra la liste originale trois fois. Lors du premier parcours, on insèrera une nouvelle fiche de TD derrière chacune des fiches de liste originale. Lors du deuxième parcours, on mettra à jour les champs `binome` des nouvelles fiches. Enfin lors du troisième et dernier parcours, on séparera les deux listes. De cette manière la fonction `copie` aura un temps de calcul proportionnel à la longueur de la liste à copier.

1)

```
new(l);
new(l^.suivant);
new(l^.suivant^.suivant);
l^.nom:='ANDRE';
l^.binome:=l^.suivant^.suivant;
l^.suivant^.binome:=nil;
l^.suivant^.nom:='BRUNO';
new(l^.suivant^.suivant^.suivant);
l^.suivant^.suivant^.nom:='CELINE';
l^.suivant^.suivant^.binome:=l;
l^.suivant^.suivant^.suivant^.nom:='DENIS';
l^.suivant^.suivant^.suivant^.binome:=nil;
l^.suivant^.suivant^.suivant^.suivant:=nil;
```

2)

```
var andre,bruno,celine,denis:pftd;
begin
  new(andre);
  new(bruno);
  new(celine);
  new(denis);
  l:=andre;
  andre^.nom:='ANDRE';
  andre^.binome:=celine;
  andre^.suivant:=bruno;
  bruno^.nom:='BRUNO';
  bruno^.binome:=nil;
  bruno^.suivant:=celine;
  celine^.nom:='CELINE';
  celine^.binome:=andre;
  celine^.suivant:=denis;
  denis^.nom:='DENIS';
  denis^.binome:=nil;
  denis^.suivant:=nil;
```

3)

```
procedure affnom(a:pftd);
begin
  if a<>nil then
  begin
    writeln(a^.nom);
    affnom(a^.suivant)
```

```
  end
end;
```

`affnom(l)` fait apparaître sur l'écran :

```
ANDRE
BRUNO
CELINE
DENIS
```

Voici une version non récursive de `affnom`.

```
procedure affnom(a:pftd);
begin
  while a<>nil do
  begin
    writeln(a^.nom);
    a:=a^.suivant
  end
end;
```

Si on intervertit les deux instructions `writeln(a^.nom)` et `affnom(a^.suivant)` alors `affnom(l)` fait apparaître sur l'écran :

```
DENIS
CELINE
BRUNO
ANDRE
```

Il est beaucoup plus difficile d'écrire une version non récursive de cette procédure.

4)

```
function nbfiches(a:pftd):integer;
begin
  if a=nil then nbfiches:=0
           else nbfiches:=1+nbfiches(a^.suivant)
end;
function nbfiches(a:pftd):integer;
var n:integer;
begin
  n:=0;
  while a<>nil do
  begin
    n:=n+1;
    a:=a^.suivant
  end;
  nbfiches:=n
end;
```

5)

```
function nbbinomes(a:pftd):integer;
var n:integer;
begin
  n:=0;
  while a<>nil do
  begin
    if a^.binome<>nil then n:=n+1;
    a:=a^.suivant
  end;
  nbbinomes:=n div 2
end;
```

6)

```
procedure affbinomes(a:pftd);
begin
  while a<>nil do
  begin
    if a^.binome<>nil then
    if a^.binome^.nom>a^.nom then
    writeln(a^.nom,'+',a^.binome^.nom);
    a:=a^.suivant
  end
end;
```

La version précédente affiche tous les binômes une fois, les deux noms étant affichés dans l'ordre alphabétique, sauf dans le cas où les deux membres du binôme ont le même nom auquel cas le binôme est affiché deux fois. La procédure suivante remédie à ce défaut. Elle modifie temporairement la liste chaînée.

```
procedure affbinomes(a:pftd);
begin
  while a<>nil do
  with a^ do
  begin
    if binome<>nil then
    if binome^.binome=nil then binome^.binome:=a else
    begin
      writeln(nom,'+',binome^.nom);
      binome:=nil
    end;
    a:=a^.suivant
  end
end;
```

7)

```
function copie(a:pftd):pftd;
var b:pftd;
begin
  if a=nil then copie:=nil else
  begin
    new(b);
    b^:=a^;
    b^.binome:=nil;
    b^.suivant:=copie(a^.suivant);
    copie:=b
  end
end;
```

8)

```
function copie2(a:pftd):pftd;
var x,y:pftd;
begin
  if a=nil then copie2:=nil else
  begin
    x:=a;
    repeat
      new(y);
      y^:=x^;
      x^.suivant:=y;
      x:=y^.suivant
    until x=nil;
    x:=a;
    repeat
      y:=x^.suivant;
      if y^.binome<>nil then
      y^.binome:=y^.binome^.suivant;
      x:=y^.suivant
    until x=nil;
    copie2:=a^.suivant;
    x:=a;
    repeat
      y:=x^.suivant;
      x^.suivant:=y^.suivant;
      x:=x^.suivant;
      if x<>nil then y^.suivant:=x^.suivant
    until x=nil
  end
end;
```