Final de NF01 automne 2006
Récursivité

Soit la fonction récursive suivante :

```
function f(n : integer):integer;

begin
   writeln(' n vaut ',n);
   if (n = 0) then
      f := 1
   else
      f := f(n+1);
end;
```

Question 1 (1 pt) : La fonction f est appelée avec n>= 0. Justifier la terminaison ou la non-terminaison de cette fonction.

Réponse : Si n est nul cette fonction retourne 1 et elle termine
Si n est strictement positif, alors il est différent de 0, donc c'est la partie 'else' qui est effectuée. Nous relançons donc l'appel récursif avec n+1, or le test d'arrêt de la récursivité s'effectue avec 0. Conclusion : nous avons une infinité d'appels récursifs dans ce cas. Cela ne termine pas. Le problème vient de la façon de relancer la récursivité, le problème à traiter ne décroît pas il croît.

Note pour toi : curieusement cela termine quand même avec mon compilateur pascal sous linux.

Explications : supposons un codage sur 1 octet pour n. Nous travaillons avec des entiers signés, donc un codage 'en complément à la base plus un'. De fait, « 00000000 » représente 0, « 01111111 » représente 127 et « 10000000 » représente –128. L'incrémentation n+1 va donc nous faire atteindre la valeur « 11111111 » puis normalement « 100000000 » sur 9 digits, si le compilateur ne génère pas un plantage pour le débordement alors le nombre considéré est « 00000000 » sur 8 digits et cela s'arrête. Je doute fort qu'ils trouvent cette explication sur laquelle j'ai du cogiter avant de trouver la réponse. Mais bon, je compterai juste s'il y en a un ou une qui me donne cette solution ….

Soit la fonction récursive suivante :

```
(* g est appelée avec n >= 0 *)
function g(n : integer):integer;

begin
   if (n <= 1) then
      g := 1
   else g := 1 + g(n-2);
end;
```

Question 2 (1 pt) : La fonction g est appelée avec n>= 0. Justifier la terminaison ou la non-terminaison de cette fonction.

Réponse :
Si n est nul, (0<=1) est vrai, cette fonction retourne 1 et elle termine. Si n vaut 1, (1<=1) est vrai, cette fonction retourne aussi 1 et elle termine. Si n est strictement supérieur à 1, alors c'est la partie 'else' qui est effectuée. Nous relançons donc l'appel récursif avec n-2, or le test d'arrêt de

la récursivité s'effectue avec (n <= 1). Conclusion : nous construisons une suite de n qui décroît et qui finalement va atteindre une valeur qui sera <= 1. Donc la fonction g termine avec n >= 0.

Soit la procédure récursive suivante :

```
(* h est appelée avec n >= 0 *)
procedure h(t : array[1..10] of integer; n : integer);

begin
   if(n >= 0) then
   begin
      writeln(n,' ',t[n]);
      h(t, n-1);
   end
end;
```

Question 3 (1 pt) :

La procédure h est appelée avec un tableau t d'au plus 10 entiers et n<=10. Justifier la terminaison ou la non-terminaison de cette procédure.

Réponse : Il va y avoir un problème.
Quand t[1] est atteint, on affiche 1 et t[1]. Puis on appelle h(t, 0) récursivement. Le test (0>=0) va être vrai, le code va chercher à afficher 0 et t[0], or en pascal la case 0 n'est pas sensé exister.

Note :
Curieusement cela fonctionne aussi avec mon compilateur car la case 0 existe mais n'est jamais utilisée bien qu'elle soit réservée. Le compilateur pascal a du être écrit avec un compilateur C, et les concepteurs n'ont pas dû se casser la tête, ils ont alloués n+1 cases dont le indices vont de 0 à n. Le programmeur pascal n'utilisant que les cases d'indices 1 à n, pas vu, pas pris.

Question 4 (1pt) : Soit la définition de type suivante :

```
type tab = array[1..10] of integer ;
```

Concevoir et écrire en pascal une fonction **somme_tableau** de deux paramètres : un tableau de ce type **tab** et un entier **n** tel que 0 < **n** <= 10. Cette fonction effectuera récursivement la somme des éléments du tableau depuis l'indice n jusque 0 et retournera cette somme.

Réponse :

```
type tab = array[1..10] of integer ;

function somme_tableau(t : tab; n : integer ):integer;

begin
   if (n=0) then
      somme_tableau := 0
   else
      somme_tableau := t[n]+ somme_tableau(t,n-1);
end;
```

Note : là, cela marche sans problèmes …

Question 5 (1pt) : soit la fonction

```
function is_mot(chaine_en_cours    : string):integer;

var ok_variable : integer;

begin
   writeln(' chaine_en_cours ', chaine_en_cours);
   if ((chaine_en_cours[1] = 'a') or (chaine_en_cours[1] = 'b')) then
   begin
      if (length(chaine_en_cours)<2) then
       is_mot := 0
      else
       if ((chaine_en_cours[2] = '+') or (chaine_en_cours[2] = '-')) then
          is_mot := 1
       else
        begin
          ok_variable := is_mot(Copy(chaine_en_cours,2,length(chaine_en_cours)-
2));
            if ((ok_variable = 1) and
            ((chaine_en_cours[length(chaine_en_cours)] = '+')
            or (chaine_en_cours[length(chaine_en_cours)] = '-'))) then
             is_mot := 1
          else
             is_mot := 0;
        end;
   end
   else
      is_mot := 0;
end;
```

La faire fonctionner avec **chaine_en_cours = ‘abc-+-‘** puis avec chaine_en_cours = ‘a++’. Que fait cette fonction récursive ?

réponse :

Elle retourne 1 si la chaîne est composé d’autant de ‘a’ ou ‘b’ que de signes ‘+’ ou ‘-‘.
Sinon elle retourne 0.

Note : C’est un des éléments des exercices sur les diagramme de Conway que l’on fait cette semaine. D’ici le final, ils auront largement oublié…