Corrigé des TD 1 à 5

1 Premier Contact

1.1 Somme des n premiers entiers

```
1  (* Somme des n premiers entiers *)
2  program somme_entiers;

4  var
     n, i, somme: integer;
6
   begin
8    (* saisie du nombre n *)
     write('Saisissez un nombre n: ');
10   readln(n);

12   (* calcul de la somme *)
     somme:=0;    (* la somme vaut 0 quand on n'y a rien ajouté *)
14   for i:=1 to n do somme:=somme+i;

16   (* affichage du résultat *)
     writeln('La somme des ', n, ' premiers entiers vaut ',somme);
18 end.
```

Lorsque n devient trop grand, on dépasse la borne supérieure du type `integer`, qui est fixée à 32767. Les résultats sont alors faux. Ce problème apparaît à partir de $n = 256$.

1.2 Table de multiplication

```
1  (* table de multiplication *)
2  program multiplication;

4  var
     ligne, colonne: integer;
6
   begin
8    for ligne:=1 to 10 do
       begin
10       for colonne:=1 to 10 do
           write(ligne*colonne:4);
12       writeln;
       end;
14 end.
```

1.3 Suite de Fibonacci

```
1  (* suite de Fibonacci *)
2  program fibonacci;

4  var
     u: integer;   (* contient u(n-2) *)
6    v: integer;   (* contient u(n-1) *)
     w: integer;   (* contient u(n)   *)
8    n, i: integer;

10 begin
     write('Combien de termes voulez-vous ? ');
12   readln(n);

14   (* initialisation *)
     u:=1; v:=1;   (* u(0) et u(1) *)
16   writeln('u(0) = 1');
     writeln('u(1) = 1');
18
     (* calcul et affichage des termes *)
20   for i:=2 to n do
       begin
22       w := u + v;   (* calcul de u(n) par récurrence *)
         writeln('u(',i,') = ',w);
24       u := v;       (* passage au *)
         v := w;       (* rang n+1   *)
26     end;
   end.
```

2 Dichotomie

2.1 Jeu du nombre secret

```
1  (* jeu du nombre secret *)
2  program jeu;

4  var
     hasard: integer;
6    essai: integer;
     tentatives: integer;
8
   begin
10   randomize;
     hasard := random(100)+1;
12   tentatives:=0;

14   repeat
       tentatives:=tentatives+1;
16     write('Entrez un nombre: ');
       readln(essai);
18     if(essai>hasard) then writeln('Trop grand.');
```

```
    if(essai<hasard) then writeln('Trop petit.');
  until essai=hasard;

  writeln('Bravo. Vous avez trouvé en ',tentatives,' essais.');
end.
```

2.2 Résolution de $f(x) = 0$

```
(* Résolution d'une équation par dichotomie *)
program dicho;

var
  a,b,c,eps: real;
  fa,fc: real;

begin
  write('Borne a: '); readln(a);
  write('Borne b: '); readln(b);
  write('Précision: '); readln(eps);

  while b-a>eps do
    begin
      c:=(a+b)/2;
      (* l'équation est codée en dur ici: x^2 - 4x + 3 *)
      fa := a*a - 4*a +3;
      fc := c*c - 4*c +3;
      if fa*fc<0 then
        b:=c
      else
        a:=c;
    end;

  writeln('A la précision ',eps,', on a: ',a,' < alpha < ',b);
end.
```

3 Tableaux

3.1 Calcul des coefficients binômiaux C_n^p

```
(* calcul des coefficients binomiaux à l'aide de la formule de Pascal *)
program cnp;

const
  MAX = 10;

var
  c: array[0..MAX, 0..MAX] of integer;
  n, p: integer;

begin
  for n:=0 to MAX do      (* parcours des lignes *)
```

```
    begin
14    for p:=0 to n do    (* parcours des colonnes *)
        begin
16        if (p=0) or (n=p) then c[n,p] := 1
                            else c[n,p] := c[n-1,p] + c[n-1,p-1];
18        write(c[n,p]:4);
        end;
20    writeln;
    end;
22 end.
```

3.2 Calcul des H_n^p

On a de façon assez claire : $H_n^p = \sum_{k=0}^{n} H_{n-k}^{p-1}$.

On en déduit alors l'algorithme, qui se fait par "couches" successives de valeurs de p. A la "couche" p, on peut exprimer tous les H_n^p en fonction des valeurs calculées à la "couche" $(p-1)$.

```
1  (* calcul des H(n,p) *)
2  program hnp;

4  const
     MAXN = 10;    (* valeur maximale de n *)
6    MAXP = 7;     (* valeur maximale de p *)

8  var
     h: array[0..MAXN, 1..MAXP] of integer;
10   n, p, k: integer;

12 begin
     (* cas de base: nombre de codes à 1 chiffre *)
14   for n:=0 to MAXN do
       begin
16       h[n, 1]:=1;
         write(1:5);
18     end;
     writeln;
20
     (* cas général *)
22   for p:=2 to MAXP do
       begin
24       for n:=0 to MAXN do
           begin
26           h[n,p]:=0;
             for k:=0 to n do h[n,p]:=h[n,p]+h[n-k,p-1];
28           write(h[n,p]:5);
           end;
30       writeln;
       end;
32 end.
```

4 Fonctions

4.1 Quelques fonctions

```
1  (* évaluation du trinôme a*x^2 + b*x + c au point x0 *)
2  function trinome(a, b, c, x0: real): real;
     begin
4      trinome:=a*x0*x0 + b*x0 + c;
     end;
6
   (* factorielle itérative *)
8  function factorielle(n: integer): integer;
     var
10     fact, k: integer;
     begin
12     fact:=1;
       for k:=2 to n do fact:=fact*k;
14     factorielle:=fact;
     end;
16
   (* factorielle récursive *)
18 function factrec(n: integer): integer;
     begin
20     if n=0 then factrec:=1
              else factrec:=n*factrec(n-1);
22   end;
```

4.2 Tours de Hanoï

La solution de ce problème est assez claire si l'on pense à raisonner de façon récursive. Si l'on suppose que l'on sait déjà déplacer des piles de $n-1$ disques, on se rend compte que l'on sait déplacer des piles de n disques.

```
1  (* résolution du problème des tours de hanoï *)
2  program hanoi;

4  (* déplacement de 'nombre' élements du piquet 'depart' vers le piquet
      'arrivee' en utilisant comme pivot le piquet 'intermediaire' *)
6  procedure mouvement(depart, arrivee, intermediaire: char; nombre: integer);
     begin
8      if nombre=0 then exit;

10     (* on déplace les (n-1) premiers éléments vers le piquet intermédiaire *)
       mouvement(depart, intermediaire, arrivee, nombre-1);
12
       (* on déplace l'élément du dessous vers le piquet d'arrivée *)
14     writeln(depart, ' => ', arrivee);

16     (* on déplace les (n-1) éléments du piquet intermédiaire vers l'arrivée *)
       mouvement(intermediaire, arrivee, depart, nombre-1);
18   end;


20
   (* programme principal de test *)
22 begin
```

```
  mouvement('a','b','c',4);
end.
```

5 Booléens

5.1 Crible d'Eratosthène

La seule astuce est de stocker l'état barré ou non-barré d'un nombre dans un tableau de booléens. L'état du nombre n est stocké à l'indice n du tableau.

```
(* crible d'Eratosthène pour la détermination des nombres premiers *)
program eratosthene;

const
  max = 100;

var
  premier: array[1..max] of boolean; (* vrai si nombre premier *)
  i, j: integer;

begin
  (* au début, tous les nombres sont candidats pour être premiers *)
  for i:=1 to max do premier[i]:=true;

  write('1; ');

  for i:=2 to max do
    begin
      if not premier[i] then continue;

      (* à ce stade, on est sûr que i est premier: on l'affiche et on barre
         ses multiples *)
      write(i,'; ');
      j:=2*i;
      while j<=max do
        begin
          premier[j]:=false;
          j:=j+i;
        end;
    end;
  writeln;
end.
```

5.2 Nombres parfaits

```
(* détermine si le nombre n est parfait *)
function parfait(n: integer): boolean;
  var
    somme, i: integer;
  begin
    somme:=0;
```

```
8       (* ajout de tous les diviseurs stricts de n *)
        for i:=1 to n div 2 do
10        if (n mod i)=0 then somme:=somme+i;

12      parfait := (somme = n);
      end;
```

Les nombres parfaits sont peu nombreux ; les premiers nombres parfaits sont 6, 28, 496, 8128.

5.3 Suite de Syracuse

```
1   (* calcul des termes de la suite de syracuse *)
2   program syracuse;

4   var
      u,k: integer;
6
    begin
8     write('u[0] = ');
      readln(u);
10    k:=0;

12    repeat
        (* calcul de u[k+1] *)
14      inc(k);
        if u mod 2 = 0 then u:=u div 2
16                     else u:=3*u+1;

18      (* affichage *)
        write('u[',k,'] = ',u,'; ');
20    until u=1; (* conjecture de Syracuse: on repasse toujours par 1 *)
    end.
```

Un exemple d'éxécution correcte est $u_0 = 13$, $u_1 = 40$, $u_2 = 20$, $u_3 = 10$, $u_4 = 5$, $u_5 = 16$, $u_6 = 8$, $u_7 = 4$, $u_8 = 2$, $u_9 = 1$.

On conjecture que pour tout terme initial u_0 donné, il existe un rang $N > 0$ tel que $u_N = 1$. Cette propriété n'est pas démontrée, mais n'a jamais été mise en défaut.

6 Ce qu'il faut retenir...

6.1 Techniques de programmation

Voici un catalogue des notions fondamentales de programmation abordées cette année, avec les références aux TD correspondants :

- **Variables et types**
  Notion de variable, types `char`, `integer` (*TD n°1*), `real` (*TD n°2*), `array` (*TD n°3*), `boolean` (*TD n°5*).
- **Opérations élémentaires**
  Affectation `:=`, comparaison, opérateurs arithmétiques, y compris `div` (division entière) et `mod` (modulo). Notez que les fonctions mathématiques standard sont disponibles en TURBO-PASCAL : `exp`, `ln`, `sin`, `cos`, `tan`...
- **Structure des programmes**
  Blocs de base : `type`, `var`, `begin`...`end`. Fonctions et procédures (*TD n°4 et 5*). Passage des arguments des fonctions par valeur et par référence (*TD n°6*).
- **Structures de contrôle**
  Condition `if`...`then`...`else` (*TD n°2*), boucle `for`...`to`...`do` (*TD n°1*), répétition `while`...`do` (*TD n°5*) ou `repeat`...`until` (*TD n°2*).
- **Procédures et fonctions de base**
  Procédures élémentaires de dialogue : `write[ln]`, `read[ln]`.
  Générateur de nombres aléatoires : `randomize` et `random()` (*TD n°2*).
- **Algorithmique**
  Calcul des termes d'une suite (*TD n°1, 3, 5*).
  Calcul polynômial (*TD n°6*).
  Résolution d'équations (*TD n°2*).
  Probabilités (*TD n°7*).

6.2 Ecriture des programmes

Les quelques conseils suivants devraient vous faciliter l'écriture des programmes :

- Pensez à l'**indentation** : il s'agit de mettre en retrait le texte de chaque bloc. La structure logique de votre programme est ainsi clairement lisible, ce qui facilite la mise au point de l'algorithme.
- N'oubliez pas de mettre des **commentaires** dans vos programmes. Ils sont délimités en TURBO PASCAL par les symboles `(*...*)`. Ils permettent de donner des explications sur des parties de programme de compréhension un peu difficile.
- Si vous ne savez pas comment démarrer un programme, commencez par réaliser calmement, "à la main", la suite d'opérations que devra effectuer ce programme. Il ne s'agira plus ensuite que de procéder à la traduction en TURBO PASCAL des opérations réalisées à la main.