Exercice 1

Écrire un programme qui affiche un tableau pour convertir les degrés Fahrenheit en degrés Celsius. Par exemple, votre programme affichera sur la colonne de gauche les nombre de 0 à 100 (de 5 en 5) correspondant aux degrés Fahrenheit, et sur la colonne de droite les valeur correspondantes en Celcius.

Rappel : $c = 5 \times (f - 32)/9$

Corrigé

Pas de difficulté. Bien noter le $\backslash t$ qui permet d'aligner sans se fatiguer et les formats d'affichage.

```
#include<stdio.h>

main(){
  float celc, fahr;

  fahr = 0.0;

  printf("Fahrenheit\tCelcius\n");

  while (fahr <=100){
    celc = 5*(fahr-32)/9;
    // Plusieurs formats d'affichage
    // En chosir un.
    // printf("%f\t%f\n", fahr, celc); // \t tabule
       printf("%8.2f\t%8.2f\n", fahr, celc);
     fahr = fahr + 5;
  }
}
```

Exercice 2

1. Tracer l'exécution du programme suivant :

```
#include <stdio.h>
main(){

  const int N=10;
  int i;
  int som;
  float harmo;
```

```
  harmo = 0.0; som = 0; i = 1; /* initialisation des variables */
  //Point d'observation 1

  while (i<=N){
    som = som + i;
    harmo = harmo + 1/(float)i;
    i++;
    // Point d'observation 2
  }
  // Point d'observation 3

  printf("La somme des %d premiers entiers est : %d\n", N, som);
  printf("La somme des %d premiers termes de la"
         " serie harmonique vaut : %f\n", N,harmo);
}
```

Corrigé

Point d'observation	i	som	harmo
1	1	0	0.0
2	2	1	1,0
2	3	3	1,5
2	4	6	1,833
2	5	10	2,083
2	6	15	2,283
2	7	21	2,450
2	8	28	2,593
2	9	36	2,718
2	10	45	2,829
2	11	55	2,929
3	11	55	2,929

Pour les matheux, un joli exercice : montrer que les sommes partielles de la série harmonique ne sont jamais des nombres entiers. Si des étudiants s'ennuient, ils peuvent y réfléchir et le programmer pour voir. Évidemment, un jour ou l'autre l'ordinateur trouvera une somme partielle "entière" à son sens (si on n'y prend pas garde).

Exercice 3

On rappelle que la suite de Fibonacci est définie par : $f_1 = f_2 = 1$ et $\forall n \geq 2$, $f_n = f_{n-1} + f_{n-2}$. On pose pour tout $n \geq 2$, $u_n = f_{n+1} f_{n-1} - f_n^2$. Écrire un programme qui calcule u_n et affiche sa valeur pour $n = 2, \ldots, a$ où a est un nombre entré par l'utilisateur.

Corrigé

Normalement, les suites à double récurrence ont été vues en cours. Donc, programmer Fibonacci est facile. Mais il y a une petite difficulté supplémentaire : on doit garder 3 termes en mémoire et non pas 2.

```
#include<stdio.h>

main(){
  int s, p, pp, i, n, u;

  printf("Entrez un entier svp : ");
  scanf("%i", &n);


  pp=1;
  p=1;
  s=2;

  i=2;

  while(i <= n){
    printf("u%i vaut %i\n", i, u);
    //printf("f%i vaut %i\n", i, p); //utile pour des tests

    pp = p; //f(n-1)
    p = s;  //f(n)
    s = p + pp; //f(n+1)
    u = pp*s - p*p; //u(n)

    i++;
  }
}
```

On remarque $u_n = (-1)^n$. Ca n'a aucun intérêt pour nous de le démontrer en TD, mais si des étudiants le demande, on a toujours l'air moins bête en sachant répondre :

On peut vérifier que $1 \times 0 - 1 = (-1)^1$ puis par récurrence : $f_{n+2}f_n - f_{n+1}^2 = (f_{n+1} + f_n)f_n - f_{n+1}^2 = f_{n+1}(f_n - f_{n+1}) + f_n^2 = -f_{n+1}f_{n-1} + f_n^2 = (-1)^{n+1}$. Ca peut aussi se montrer en utilisant la relation bien connue :

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

On notera que le programme est étonnamment stable pour les grands nombres. Même quand on dépasse de loin les capacité de calcul de Fibonacci, le calcul de u_n reste correct. Parce que la formule de $u_n = (-1)^n$ reste vrai avec des nombres de Fibonacci calculés modulo un grand entier.

Exercice 4

1. Écrire un programme qui demande 2 entiers x, y à l'utilisateur et qui affiche en retour un rectangle de x lignes et y colonnes. Par exemple, si l'utilisateur choisit $x = 5, y = 3$ le programme devra afficher :

```
xxx
```

```
xxx
xxx
xxx
xxx
```

2. modifier le programme précédent. Cette fois, un seul nombre x est demandé et le programme affiche un triangle de x lignes comme suit :

```
x
xx
xxx
xxxx
xxxxx
```

3. Dernière modification, le programme doit afficher le triangle ci-dessous pour $x = 5$:

```
    x
   xxx
  xxxxx
 xxxxxxx
xxxxxxxxx
```

Corrigé

Pas de difficulté spéciale. C'est la première fois qu'on demande aux étudiants d'emboiter deux boucles.

```
#include<stdio.h>

main(){
  int x, y, i, j;

  printf("Entrez x svp : ");
  scanf("%i", &x);
  printf("Entrez  y svp : ");
  scanf("%i", &y);

  for(i=1; i<=x; i++){
    for(j=1; j<=y; j++){
    putchar('x');
    }
    putchar('\n');
  }

  putchar('\n');

  for(i=1; i<=x; i++){
```

```
    for(j=1; j<=x-i; j++){
    putchar(' ');
    }
    for(j=1; j<=2*i-1; j++){
    putchar('x');
    }
    putchar('\n');
  }
}
```

Exercice 5

1. Programmer l'algorithme d'Euclide vu en cours (pour calculer le PGCD). Le programme doit en plus afficher le nombre de soustractions nécessaires pour parvenir au résultat.

2. On propose l'algorithme suivant pour le calcul du pgcd de a et b.

$r_0 = |a|$
$r_1 = |b|$
Pour tout n, si $r_{n-1} \neq 0$, alors on définit r_n par :
$r_n = |r_{n-2} - r_{n-1}|$

On peut montrer que le dernier terme non nul de la suite (r_n) est le PGCD de a et b.

Programmer cet algorithme. Le programme doit en plus afficher le nombre de soustractions nécessaires pour parvenir au résultat.

3. Comparer les deux algorithmes en terme de nombre d'itérations dans la boucle.

Corrigé

On remarque que l'algorithme par soustraction est beaucoup moins performant. C'est l'occasion de dire qu'il y a de bons et de mauvais algorithmes.

Le if n'a pas encore été vu, donc on ne peut pas calculer la valeur absolu (ou alors avec des ruses affreuses avec while, qu'il ne faut surtout pas montrer si on veut que les étudiants fassent bien la différence en boucle et test). Le plus simple est d'utiliser la bibliothèque math.h. (mon intention en TD, c'est de dire d'utiliser abs, puis que les étudiants voient planter la compilation, puis enfin de dire d'inclure ¡math.h¿).

Une remarque sur la programmation d'abs : peut-on programmer abs avec seulement les opération +, *, -, / ? La réponse est non, car abs n'est pas dérivable en 0, et tout ce qu'on fera de partout défini avec +, *, -, / le sera. C'est intéressant de voir que de l'analyse montre qu'un langage informatique restreint a des limites.

```
#include<stdio.h>
#include<math.h>

main(){
  int a, b, s, i, sa, sb;
```

```
  printf("Entrez a svp : ");
  scanf("%i", &sa);
  printf("Entrez  b svp : ");
  scanf("%i", &sb);

  /*ALGORITHME PAR SOUSTRACTIONS*/

  a=abs(sa); b=abs(sb);
  i=0;

  while(b!=0){
    s=a; //s comme sauvegarde
    a=b;
    b=abs(s-b);
    i++;
 }

  printf("Le PGCD de a et b vaut %i.\n", a);
  printf("L'algorithme par soustractions a necessite %i soustractions.\n", i);


  /* EUCLIDE */

  a=abs(sa); b=abs(sb);
  i=0;

  while(b!=0){
    s=a; //s comme sauvegarde
    a=b;
    b=s%b;
    i++;
 }

  printf("Le PGCD de a et b vaut %i.\n", a);
  printf("L'algorithme d'Euclide a necessite %i divisions.\n", i);
}
```