Informatique S1 – Programmation C
Exercices - Corrigés

TD 4 : Les boucles *while* et *do while*

Dans ce TD, nous allons réaliser des exercices couvrant l'usage des boucles *while* et *do while* dans le langage C.

Objectifs : réaliser des boucles simples (pas de boucles imbriquées), renforcer l'importance de l'indentation.

- Réaliser des boucles simples : conversion C°→F°, calcul de x^y, factoriel …
- Tracer sur papier un programme avec boucles
- Faire la différence entre l'opérateur d'attribution « = » et l'opérateur logique « == »
- Renforcer l'importance de l'indentation
- Pour les étudiants plus avancés : série d'Euler ($1/1^2 + 1/2^2 + … + 1/k^2 + … = \pi^2/6$)

Exercice 1

a) Le programme compte le nombre de fois que l'utilisateur entre un entier positif. Le programme contient un `do-while` à l'intérieur duquel se trouve ladite ligne (« `count = count + 1 ;` »). Puisqu'il s'agit d'un `do-while`, elle sera exécutée au *minimum une fois* et au *maximum* autant de fois que souhaite l'utilisateur (jusqu'à qu'il entre « 0 » ou un entier négatif).

b) Un exercice simple pour faire la différence entre la boucle ***while*** et la boucle ***do while***.

NOTE : plusieurs étudiants m'ont posé en cours la question de la différence entre l'usage du « %d » et du « %i » dans un scanf. Cet exercice est l'opportunité pour eux de trouver la réponse par eux-mêmes (il n'y a pas de différence).

```c
#include <stdio.h>

/*  TD 4 : boucle While    */

int main () {
    int n = 1; /* ATTENTION : il faut que la valeur initiale de n ne soit pas 0
                              sinon, on ne va pas entrer dans le while */
    int count = 0;

    while (n>0) {
        printf ("Donner un entier > 0 : ");
        scanf ("%i", &n);

        printf ("Vous avez fourni %i \n", n);

        count = count + 1;
    }

    printf ("Vous avez entre %d n. entiers positifs \n", count);
}
```

Exercice 2

L'objectif de cet exercice est de rendre compte aux étudiants les différences entre while et do-while. Il faut souligner le fait qu'une boucle do-while exécute toujours au moins une fois.

Exercice 3

a) Dans cet exercice, un code est très similaire à celui vu en cours est présenté, et les étudiants doivent remplir les lacunes dans le code. Parmi ces lacunes, il y a notamment le test utilisé dans la boucle while. Ce test est légèrement différent de celui vu en cours (i<=y au lieu de i<y), l'objectif étant de leur faire réfléchir au test.

RAPPEL : on n'a pas encore vu les opérateurs d'incrément ++ et --en cours.

```
#include <stdio.h>

int main () {
   float x, y; /* valeur fournis par l'utilisateur
   float _z_;     /* z = x puissance y */
   int i;

   /* lecture des variables */
   _printf_ ("Entrez x : ");
   scanf ("_%f_", &x);
   printf ("Entrez y : ");
   scanf ("%f", _&y_);

   z = 1;
   i = 1;

   /* on multiplie x y-fois */
   while (_i <= y_) {
        z = z * x;
        i = _i + 1_;          /*
   }

   /* presentation des resultats */
   printf ("x ^ y = %f \n", _z_);
}
```

Les étudiants doivent observer que les variables utilisées dans le code sont x,y et **z**.

Les étudiants doivent observer les types des variables (float) afin de trouver la bonne correspondance (**%f**). Ils doivent aussi trouver la bonne variable à lire (**y**), en observant le code source et l'énoncé de l'exercice.

Les étudiants doivent trouver le bon test : si i=1, on doit avoir i<=y et non i<y comme vu en cours.

Il ne faut pas oublier l'incrément de la variable i qui contrôle le loop.

On identifie dans le code quelle variable continent le résultat attendu (x^y)

b) L'objectif de la « question défi » est d'inciter les étudiants à aller chercher des informations complémentaires sur le langage C et les bibliothèques. Le même code peut être réalisé à l'aide de la fonction « pow » dans la bibliothèque math.h (code ci-dessous). Les fonctions les plus populaires de la bibliothèque math.h seront discutés dans la prochaine séance de CM, avec la boucle for.

ATTENTION : depuis quelques versions, le compilateur gcc sous Linux demande que la bibliothèque math.h soit 'linkée' au programme de manière dynamique, à l'aide de l'option « -lm ». Donc, pour compiler le code ci-dessous il faut utiliser la ligne « gcc **–lm** –o pow pow.c ».

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

/*  TD 4 : Question 3b
 *  Calculer x^y sans utiliser une boucle, a l'aide des
 *  bibliotheques de fonction.
 *
 * ATTENTION : Compiler avec -lm sous linux
 */

int main () {
   float x, y;
   float z;

   /* lecture des variables */
```

```
   printf ("Entrez x : ");
   scanf ("%f", &x);
   printf ("Entrez y : ");
   scanf ("%f", &y);

   z = pow (x,y);

   /* presentation des resultats */
   printf ("x ^ y = %f \n", z);

}
```

Exercice 4

a) Comme l'exercice précédent.

```
#include <stdio.h>

/*  TD 4 : Factoriel de n
 *  Factoriel de n = 1 * 2 * ... * n-1 * n
 */

int main () {
  int n;
  int i = 1;
  int fact = 1;  /* factorie

  printf ("Entrez n : ");
  scanf ("%d", &n);

  do {
       fact = fact * i;
       i = i + 1;
  } while ( i <= n );

  printf ("Factoriel de %d est %d \n", n, fact);
}
```

Ils doivent absolument attribuer une valeur initiale à i et à fact s'ils veulent que le programme fonctionne correctement. Par ailleurs, ils doivent trouver la bonne valeur (1 et non 0), en fonction de la boucle do-while et de ses instructions.

Pour la 1ère lacune, il faut réfléchir à la logique du programme (un factoriel).

Pour la 2nd, il ne faut pas oublier l'incrément de la variable i qui contrôle le loop.

b) Conversion entre do-while et while.

```
#include <stdio.h>

/*  TD 4 : Factoriel de n
 *  Factoriel de n = 1 * 2 * ... * n-1 * n
 */

int main () {
  int n;
  int i = 1;
  int fact = 1;  /* factoriel de n */

  printf ("Entrez n : ");
  scanf ("%d", &n);

  while ( i <= n ) {
       fact = fact * i;
       i = i + 1;
  }

  printf ("Factoriel de %d est %d \n", n, fact);
}
```

Exercice 5

Cet exercice est une extension de l'exercice 2, puisqu'on utilise la somme d'un ensemble pour calculer sa moyenne. Cependant, il ne faut pas négliger sa difficulté : plusieurs étudiants auront du mal à faire l'abstraction nécessaire de l'exercice 2 pour arriver à la solution de celui-ci.

```
#include <stdio.h>

/*  TD 4 : calculer la moyenne d'un exnemble de 10 numeros réels  */

int main () {
  float xi=0;
  int n=0;
  float somme = 0.0;
  float moy = 0.0;

  while (n<10) {

      printf ("Entre un numero reel (x%d): ", n);
      scanf ("%f", &xi);

      somme = somme + xi;
      n = n + 1;
  }

  moy = somme / 10;  //ou moy = somme / n;

  printf ("La moyenne est : %f \n", moy);
}
```

Exercice 6

Comme dans l'exercice précédent, les étudiants doivent désormais écrire leur propre code : identifier les variables nécessaires, les déclarer, trouver les instructions et les structures de contrôle nécessaires, etc.

```
#include<stdio.h>
/* Source : Poly TD de N.Trotignon */
int 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 7

L'objectif de cet exercice est de leur conscientiser à l'importance d'une bonne indentation. Les deux codes sources sont exactement égaux et syntaxiquement correctes, sauf que le premier est totalement illisible. Tous les deux calculent et affichent x^x et sqrt (x). La trace d'exécution pour le second (en considérant que l'utilisateur a fourni la valeur 4) est la suivante :

	x	i	xpow	racx
Point d'observation 1	?	?	?	?
Point d'observation 2	4	0	1	?
Point d'observation 3	4	1	16	?
Point d'observation 3	4	2	64	?
Point d'observation 3	4	3	256	?
Point d'observation 5	4	4	256	2

Exercice 8

Exercice sans difficulté particulière.

	x	racine
Point d'observation 1	?	?
Point d'observation 2	4	?
Point d'observation 2	1	2
Point d'observation 2	0	1
Point d'observation 3	0	0

Exercice 9

Au-delà du fait que le programme calcule une série de Fibonacci, et que pour cela, elle fait l'échange des valeurs des variables (u0=u1 et u1=un), ce qui peut confondre certains étudiants, le programme comporte un seul erreur : dans le teste du do … while (op = 0) au lieu de do … while (op == 0). Cet exercice est donc une occasion de rappeler aux étudiants la différence entre l'opérateur d'affectation (=) et celui d'égalité (==).

Exercice 10

Exercice pour amuser les étudiants avancés :

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

int main (int argc, char argv[]) {
 long int k = 0;
 long int i = 1;
 double euler = 0.0;
 double diff = 0.0;

 printf ("Entrez une valuer pour k: ");
 scanf ("%ld", &k);
 printf ("Calcul de la Serie d'Euleur pour %ld: ", k);

 while (i<=k) {
     euler += 1/pow(i,2);
     i++;
 }
 /* la serie d'euleur doit s'approcher de la valeur de PI^2/6 */
 diff = (pow(M_PI,2)/6) - euler;
 printf ("%lf ~ %lf (%le)\n", euler, (pow(M_PI,2)/6), diff);
 return (0);
}
```