Informatique S1 – Programmation C
Exercices Corrigés

TD 2 : Structure générale d'un programme C

Exercice 1

a) Le programme présente 5 erreurs, à savoir :
   a. Manque d'un « { » après la déclaration « main ( ) »
   b. Manque d'un « ; » au premier « printf »
   c. Manque le « & » au « scanf »
   d. Les variables « myInt1 » et « myInt2 » sont utilisées « myint1 » et « myint2 » (« i » au lieu de « l »)
   e. Il manque une « , » au second « printf »

   L'objectif ici est de rappeler aux étudiants l'importance des détails dans la programmation C. Le compilateur ne leur dira pas toujours de manière claire l'origine de l'erreur, et certains erreurs passeront même inaperçues par le compilateur. On doit être attentif aux détails pour éviter les problèmes. L'exemple attire l'attention sur :

   - l'ouverture et fermeture des blocs (suggestion : les conseiller à écrire toujours « { } » ensemble, et ajouter le contenu entre les deux, pour ne rien oublier)
   - les « ; » sont toujours obligatoires
   - les lettres majuscules et minuscules sont différentes (i ≠ l)
   - le « & » dans le scanf est nécessaire
   - les « , » pour séparer les éléments dans le printf sont aussi nécessaires

   A travers cet exercice, on peut également souligner une déclaration alternative pour le main, sans les indications de retour et des paramètres (juste « main ( ) ») est possible.

b) L'objectif de cet exercice est de les faire comprendre le programme pour qu'ils puissent mieux voir les problèmes et les corriger ensuite. C'est aussi une première opportunité de faire comprendre l'utilité des commentaires et des noms de variables significatifs : comme il n'y a pas de commentaires et les variables se nomment « myInt », c'est moins évident que le programme fait une conversion d'un nombre d'heures en secondes.

c) Voici le code corrigé :

```
#include <stdio.h>

main () {
  int myInt1, myInt2;

  printf ("Entrer le nombre d'heures : ") ;
  scanf ("%d", &myInt1);

  myInt2 = myInt1 * 3600;

  printf ("Il y a %d s en %d h ", myInt1, myInt2);
}
```

Exercice 2

Exercice simple dont l'objectif est que les étudiants puissent faire par eux-mêmes leur premier programme. C'est sensiblement la même chose que les exemples vu en cours et dans l'exercice précédent, mais ils doivent le faire eux-mêmes.

```
#include <stdio.h>
main () {
 int nombre;
 printf ("Bonjour, entrer un nombre entier s'il vous plait :");
 scanf ("%d", &nombre);
 printf ("Le nombre que vous avez entre est %d\n", nombre);
}
```

Exercice 3

Tel que l'exercice précédent, celui-ci demande également aux étudiants d'écrire pour eux-mêmes un programme qui calcule le double et le triple d'un nombre entier. Idéalement, ils doivent le faire par étape (d'abord le double, ensuite le triple), pour bien fixer les principes. Pour les étudiants plus avancés, on peut leur demander de faire l'exercice avec un nombre minimal d'instructions.

```
/*
  Écrire un programme en langage C qui lit un nombre entier fourni par
  l'utilisateur, calcule et affiche à l'utilisateur le double de ce nombre.
  Comment faire pour que le programme affiche également le triple ?
*/
#include <stdio.h>
main () {
 int n;

 //d'abord, en utilisant des variables pour le double et triple
 int n2, n3;
 printf ("Entrer un nombre entier : ");
 scanf ("%d", &n);
 n2 = n * n;
 printf ("Le double de %d est %d\n", n, n2);
 n3 = n2 * n;
 printf ("Le triple de %d est %d\n", n, n3);

 //sans variables autres que n
 printf ("Le double de %d est %d et le triple est %d\n",
             n, (n*n), (n*n*n));
}
```

Exercice 4

Exercice 4a à 4c : pas de soucis particulier, juste application des opérateurs arithmétiques (corrigé ci-dessous).

Exercice 4d : Les étudiants auront vu en cours que les opérateurs dépendent du type des variables (le / entre deux entiers n'est pas pareil que entre deux float). Cet exercice doit illustrer cet aspect. Il s'agit aussi de les préparer pour le typecast, qui sera vu dans la séance suivante.

Exercice 4a : Les étudiants auront vu en cours que chaque type à une taille, ce qui implique une limite. Cet exercice doit montrer ces limites : la somme, au lieu de donner une réponse correcte, donne un numéro négatif. À la fin du semestre, lors de la séance sur les systèmes de numérotation, on retournera sur ce point, et notamment sur le fait qu'on utilise un bit pour représenter le signal dans un entier.

```
/*
 * a) Écrire un programme qui demande à l'utilisateur deux nombres entiers et
 *    qui affiche la somme entre les deux nombres.
 * b) Étendre le programme précédent pour qu'il affiche également la
 *    soustraction entre les deux nombres lus (premier – second).
 * c) Étendre encore le programme précédent pour qu'il affiche la division
 *    (premier / second) et le reste de la division (premier % second).
 * d) À partir de la dernière version du programme précédent, que se
 *    passe-t-il si l'utilisateur introduit les nombres « 9 » et « 2 »
 *    respectivement ? Quelles réponses aurons-nous pour la division et
 *    le reste ? Comment faire pour afficher le résultat de la division
 *    lorsque ce résultat n'est pas un nombre entier ?
*/

#include <stdio.h>


main ()
{
    int a, b;
    float r;
    a = 0;
    b = 0;
    r = 0.0;


    printf ("Entrer a: ");
    scanf ("%d",&a);

    printf ("Entrer b: ");
    scanf ("%d", &b);

    //exercice 4a: a+b
    printf ("a + b = %d\n",(a+b));

    //exercice 4b: a-b
    printf ("a - b = %d\n",(a-b));

    //exercice 4c: a/b et a%b
    printf ("a / b = %d \t a%%b = %d \n",(a/b),(a%b));

   //exercice 4d: float a/b
   // juste (a/b) realise la division entiere, meme avec le typecast
   // printf ("a / b = %f (sans typecast) \n", (float)(a/b));

   // pour en être sur, il faut passer par une variable float
   // et utiliser le typecast
    r = (float) a/b;
    printf ("a / b = %f \n", r);
}
```

Exercice 5

L'objectif de cet exercice est d'initier les étudiants au « traçage » d'un programme. Le fait de pouvoir tracer un programme sur papier leur sera très utile pour y trouver les éventuelles erreurs cachées. En suivant la méthode proposée par Nicolas dans les années précédentes, tracer l'exécution d'un programme, c'est donner les valeurs de toutes les variables à chaque point d'observation. Quand on trace un programme, on ne se soucie pas de ce qui est affiché par les printf, juste des valeurs des variables. Afin d'uniformiser tout ça et de se préparer à des exercices plus difficiles, il souhaitable que les étudiants adoptent la présentation suivante :

Point d'observation	variable 1	variable 2	…	variable n
1	*valeur*	*valeur*	…	*valeur*
2	*valeur*	*valeur*	…	*valeur*
…	…	…	…	…
i	*valeur*	*valeur*	…	*valeur*

a) La trace l'exécution du programme est indiquée ci-dessous. À noter que la valeur de « celc » est fournie par l'utilisateur. Lors du traçage, les étudiants doivent supposer que l'utilisateur a fourni une valeur donnée et l'utilisée. À la fin de l'exécution (point d'observation 3), la value de la variable « fahr » est 50 si l'utilisateur a fourni la valeur « 10 ».

Point d'observation	celc	fahr
1	0.0	0.0
2	10 (donné par l'utilisateur)	0.0
3	10	50

b) L'objectif ici est de leur rappeler qu'on doit indiquer « le bon format » au printf et au scanf. Ils auront vu en cours quelques formats (%d,%i,%f,%e,%c) et devront pouvoir répondre aisément à cette question : %f est pour afficher le contenu d'une variable de type float, %d sert à afficher une variable de type int.

Exercice 6

L'objectif de cet exercice n'est pas de présenter aux étudiants la suite de Fibonacci, mais surtout de les entrainer aux traces, qui lui seront très utiles lors du debug de leurs propres programmes.

Le second objectif de cet exercice est de montrer l'importance de l'initialisation des variables : lorsqu'on déclare une variable, elle peut avoir n'importe quelle valeur (et surtout pas zero). Il faut toujours initialiser les variables avant de les utiliser afin de s'assurer qu'elles auront la valeur attendue. On peut demander aux étudiants d'exécuter à plusieurs reprises le programme (une foi qu'ils l'auront tapé), afin de voir que les différents valeurs qui pourront s'afficher lors du premier printf.

Enfin, cet exercice montre bien comme peut être difficile de comprendre un programme dont on n'est pas l'auteur et qui n'a pas beaucoup de commentaires, d'où l'importance de bien utiliser les commentaires.

Trace :

Point d'observation	p	s	n
1	Indéterminé	Indéterminé	Indéterminé
2	1	1	Indéterminé
3	1	2	1
4	2	3	2
5	3	5	3
6	5	8	5
7	8	13	8

Exercice 7

a) Le programme « echange1.c » illustré ne fait pas l'échange des valeurs entre deux variables. La valeur de la variable a est perdue, comme le montre la trace du programme.

Point d'observation	a	b
1	3	5
2	5	5

b) La solution la plus pratique pour la question de l'échange est l'usage d'une variable tampon. Pour les étudiants les plus avancés, on peut leur proposer d'étendre la solution à la permutation des trois valeurs (c'est-à-dire qu'après l'exécution du programme, « a » doit avoir la valeur de « b », « b » celle de « c » et « c » celle de « a »).

```
#include<stdio.h>

main()
{
    int a, b;
    int t; //Variable temporaire
    a=3;
    b=5;
    printf("a vaut %i\n", a);
    printf("b vaut %i\n", b);
    t=a; //On sauvegarde la valeur de a dans la variable temporaire
    a=b;
    b=t; //On recupere la valeur de a
    printf("a vaut %i\n", a);
    printf("b vaut %i\n", b);
}
```