Introduction à la Programmation C
Corrigés des Exercices

Université Paris 1 – Panthéon Sorbonne
L1 – MASS
Informatique S1

Informatique S1 – Programmation C
Exercices - Corrigés

TD 1 : Prise en main de l'environnement

Solution Exercice 1

Objectif : prise en main de l'environnement graphique et de l'éditeur *gedit.*

L'idée est d'inviter les étudiants à l'usage du système Linux, en le démystifiant. Ils doivent prendre en main l'éditeur *gedit*, se familiariser avec lui et avec le mode de coloration. On doit s'assurer qu'ils enregistrent leur fichier helloworld.c dans le répertoire home de l'utilisateur.

Dans la question sur l'usage du navigateur, profiter pour rappeler l'importance de l'usage de l'adresse de courrier électronique de l'université. S'assurer que les étudiants savent envoyer un ficher en pièce jointe par le webmail de l'université (ça peut nous servir pour la suite).

Exercice 2

Objectif : initier les étudiants à l'usage de la ligne de commande et à l'utilisation des commandes simples.

a) Menu applications -> Accessoire -> terminal

b) Pour trouver le répertoire courant, utiliser la commande « pwd ». Le chemin complet doit être quelque chose comme « /home/user ».

c) Afficher le contenu avec la commande « ls »

d) Commande : « mkdir code »

e) Commandes : « cd code » suivi de « ls –la ». Comme le répertoire est vide, ls –la n'indique que « . » et « .. ». Expliquer (à l'aide du guide de référence) le significat de chaque.

f) Utiliser la commande « pwd » pour voir le chemin complet.

g) Faire observer l'usage du « .. » pour référencer le répertoire précédent.

h) L'objectif de cet exercice est d'indiquer aux étudiants que les lettres majuscules et minuscules ne sont pas pareilles sous Linux. S'ils ont bien enregistré le fichier helloword.c avec ce nom et sur le home de l'utilisateur, la première commande ne doit pas marcher, tandis que la deuxième doit afficher le contenu du fichier.

i) Commande : cp helloworld.c HelloWorld.c

j) Maintenant qu'on a crée un fichier « HelloWorld.c », la commande cat de la question h doit marcher. Renforcer le fait que les lettres majuscules et les minuscules sont différentes.

k) Commande : mv Helloworld.c /home/user/code (ça peut varier selon le répertoire)

l) Exercice simple pour faire comprendre aux étudiants qu'un répertoire doit être vide avant de le supprimer. (conseil : éviter quand-même de les indiquer l'usage de « rm –rf » dans un premier moment, ça cause de dégâts facilement)

Exercice 3

a) L'usage de l'USB sur Linux est toujours un peu délicat. L'objectif ici est permettre aux étudiants de le prendre en main (pour que dans les séances suivantes, ils puissent enregistrer sur leur clé USB leurs programmes). Si le groupe paraît avancé, il existe la possibilité de leur expliquer comment Linux voit les dispositifs (tout est un fichier sur Linux).

b) Commande : cp –r code /media/USB\ DISK (ça peut varier selon le *moint point*)

c) L'objectif ici est d'indiquer qu'il faut déconnecter la clé USB (tel qu'on le fait sous Windows). On peut le faire à l'aide de l'interface graphique (clique droit sur l'icône qui normalement doit apparaître au bureau, option « démonter le volume ») ou avec la commande « umount /media/USB\ DISK » (ça peut varier en fonction du *moint point*). Attention : l'étudiant ne peut pas avoir d'application ouverte (éditeur de texte, terminal…) utilisant la clé pour qu'elle soit déconnectée.

Exercice 4

a) La commande « which » indique le chemin utilisé lors de l'exécution d'une commande donnée, tandis que la commande « whereis » indique le chemin où on peut trouver cette commande (éventuellement les répertoires qui lui font référence).

b) L'objectif de cet exercice est de le faire comprendre la redirection de la sortie avec « > » et « >> ».

c) L'objectif ici est de présenter l'usage du « \ » pour pouvoir écrire une commande sur plusieurs lignes. Deuxième objectif est de le faire consulter le guide de référence, dans lequel il y a un exemple de « \ » qui utilise la commande « cat » (en somme, la solution de l'exercice est là pour ceux qui voudront bien lire le guide).

   Par ailleurs, pour que le prochain exercice soit clair, il faut qu'ils exécutent la redirection (>>) sur le ficher « test_redir.txt » au moins trois fois, afin d'avoir suffisant de contenu pour que l'exécution des commandes « more » et « less » soit sensiblement différente de celle de « cat ».

d) L'idée ici est le montrer que « more » et « less » présente un fichier page par page, contrairement à « cat » que le fait défiler sans arrêt à l'écran. Attention : les commandes « more » et « less » ne sont pas toujours disponibles (parfois seulement un ou l'autre est disponible).

e) Exemple de redirection avec le « | » (*pipeline*). Les étudiant doivent comprendre qu'on peut exécuter deux commandes de suite, en faisant la redirection de la sortie du premier (le texte afficher par cat) vers l'entrée du second (le more/less) avec le pipeline.

f) L'objectif ici est de montrer la commande man comme une aide importante.

g) L'idée ici est de montrer l'usage du joker « * » (*mask*).

h) Usage de la commande « zip » pour la création d'archives. Faire remarquer aux étudiants qu'ils peuvent envoyer par mail les programmes qu'ils auront crée pendant les séances de TD plus facilement s'ils envoient un seul fichier zip (au lieu de plusieurs fichiers .c).

i) L'objectif ici est qu'ils puissent voir une autre fonctionnalité utile de gedit (menu Rechercher→Aller à la ligne) et de préparer l'exercice suivant.

j) Les étudiants doivent utiliser l'option « -l » avec la commande « unzip » (exemple sur le guide de référence) pour éviter d'écraser leurs fichiers. Il faut qu'ils soient conscients que la commande « unzip » ne va pas poser des questions avant d'écraser un fichier.

Exercice 5

L'objectif de cet exercice est d'abord de montrer les effets de la compilation : la création d'un fichier binaire « a.out » dans le premier cas, « helloworld » dans le second. C'est également l'opportunité de leur parler que les programmes ne s'appellent pas forcément « quelquechose.exe » sous Linux, qu'ils peuvent avoir n'importe quel nom.

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);
}
```

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);
}
```

Informatique S1 – Programmation C
Exercices - Corrigés

TD 5 : Les boucles *for*

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

Objectifs : présentation des opérateurs unaires ++, --, +=, -=, des boucles *for* et des boucles imbriquées.

- Opérateurs unaires ++, --, +=, -=
- Présentation boucle for
- Présentation boucles imbriquées
- Concepts complémentaires
  - Fonction de bibliothèque : getchar, putchar

Exercice 1 – Déclaration for

L'objectif de l'exercice est d'attirer l'attention vers la définition de la boucle for : *for (attribution ; test ; incrément) instructions*. On peut faire plusieurs attributions (séparées par des « , ») et différents incréments (aussi séparés par les « , ») dans les parties leur correspondant. L'exercice renforce la possibilité d'utilisés plusieurs attributions, et la possibilité d'utiliser les opérateurs unaires (« ++ ») pour l'incrément.

```
#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);

   /* on multiplie x y-fois */
   for (i=0, z=1; i < y; i++) {
        z = z * x;
   }

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

Exercice 2 – while-for

Exercice essentiellement de traduction de while vers for, étant donné que la semaine dernière, ils ont fait en cours le même programme avec while et do-while.

```
#include <stdio.h>

/*  TD 5 : Factoriel de n
 *  Factoriel de n = 1 * 2 * ... * n-1 * n
 */

int main () {
```

```
  int n;
  int i;
  int fact;  /* factoriel de n */

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

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

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

Exercice 3 – while-for

Idem l'exercice antérieur.

```
#include <stdio.h>

/*  TD 5 : calculer la moyenne d'yun exnemble de 10 numeros réels  */

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

  for (n=0; n<10; n++) {
      printf ("Entre un numero reel (x%d): ", n);
      scanf ("%f", &xi);
      somme = somme + xi;
  }

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

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

Exercice 4

Petit exercice rigole et facile qui fait un simple compte à rebours. Le détail ici est que le compte à rebours est inversé : on compte de i=n à i=0. Il nous faut utiliser donc for (i=n ; i >0 ; i--) au lieu de la traditionnel boucle for(i=0 ; i<n ; i++).

```
#include <stdio.h>
/*  TD 5 : Ecrivez un programme qui réalise un compte à rebours
    avant de se terminer. */
int main ()
{
   int n = 0;
   int i=0;
   printf ("Entrer un n. entier : ");
   scanf ("%d", &n);

   for (i=n; i>0 ; i--)
      printf ("... %d ...\n",i);

   printf ("!!!! BOOOOOMMMMMM !!!\n");

}
```

Exercice 6 – while-for

a) L'exercice demande d'écrire un programme capable d'afficher un tableau avec la conversion °C - °F de -15°C à +45°C. Cet exercice est l'opportunité de souligner deux faits : (i) la variable de contrôle du for n'est pas forcément un int (dans ce cas, il peut s'agir d'un float) ; (ii) l'attribution initiale du for n'est pas forcément « i=0 » ou « i=1 », c'est n'importe quelle valeur (dans ce cas, celc=-15).

```
#include <stdio.h>

/* Conversion C-F */
main () {
  float celc, fahr;

  printf ("Celsius\tFahrenheit \n");

  for (celc=-15; celc <= 45; celc++) {
      fahr = (celc*9)/5 + 32;
      printf (" %.1f \t %.2f\n", celc, fahr);
  }

}
```

b) Dans cet exercice, on demande aux étudiants d'afficher le même tableau, de 5 en 5 degrés. C'est l'opportunité de leur signaler que l'incrément n'est pas forcément « i++ », mais une affectation comme n'importe quelle autre (ici, on utilise celc = celc + 5).

```
#include <stdio.h>

/* Conversion C-F */
main () {
  float celc, fahr;

  printf ("Celsius\tFahrenheit \n");

  for (celc=-15; celc <= 45; celc=celc+5) {
      fahr = (celc*9)/5 + 32;
      printf (" %.1f \t %.2f\n", celc, fahr);
  }

}
```

Exercice 5 – Bloc d'instructions

L'objectif de cet exercice est de montre l'importance de l'indentation pour la visibilité du code source. Deux codes sont présentés. Le premier affiche correctement le carré et le cube des petits entiers, grâce à une boucle for. Le second affiche seulement le carré et le cube du nombre 5 à cause de la boucle for, qui ne compte pas un bloc définie par « { } » et qui donc contient uniquement la ligne « carre = a*a; ».

a) L'exécution du code A affiche les carrés et les cubes des nombres 1 à 5. Le code B affiche uniquement la ligne « 6 25 150 ». Leurs traces sont les suivants :

Code A	a	carre	cube
Pont d'observation 1	?	?	?
Pont d'observation 2	1	1	1
Pont d'observation 2	2	4	8
Pont d'observation 2	3	9	27

Pont d'observation 2	4	16	64
Pont d'observation 2	5	25	125
Pont d'observation 3	6	25	125

Code B	a	carre	cube
Pont d'observation 1	?	?	?
Pont d'observation 2	1	1	?
Pont d'observation 2	2	4	?
Pont d'observation 2	3	9	?
Pont d'observation 2	4	16	?
Pont d'observation 2	5	25	?
Pont d'observation 3	6	25	150

b) Différence entre les deux codes : code A contient une boucle for qui définit un bloc de 5 instructions, tandis que dans le code B la même boucle ne définit pas un bloc, exécutant uniquement une instruction

c) Le code A correspond mieux aux spécifications énoncées dans les commentaires.

Exercice 7 – Boucles imbriquées

a) Le programme affiche la table de multiplication de 1 à 10. Les trous dans le code correspondent uniquement aux boucles for. L'objectif est de montrer aux étudiants l'usage des boucles imbriqués.

```
#include <stdio.h>

int main () {
  int i, j ;
  int max = 10;

  /* affiche l'entete */
  printf ("      I");
  for (i=1; i<=max; i++)
     printf ("%4d", i);

  printf ("\n");
  printf ("--------");

  for (i=0; i<max; i++)
     printf ("----");

  printf ("\n");

  /* affiche les differents lignes */
  for (i=1; i<=max; i++) {
     printf ("%4d  I", i);
     for (j=1; j<=max; j++) {
         printf ("%4d", i*j);
     }
     printf ("\n");
  }
}
```

b) Pour afficher une table de multiplication jusqu'au numéro 15, il faut tout simplement modifier la valeur de la variable « max ». Rappel : les étudiants n'ont pas encore vu les « #define », cet exercice constitue une préparation pour la définition des constantes.

Exercice 8 – boucles imbriquées

```
#include<stdio.h>

/** TD 5: afficher des retangles et triangles
 * Source : poly de M. Troignon
 */

main()
{
    int x, y, i, j;
    printf("Entrez x svp : ");
    scanf("%i", &x);
    printf("Entrez y svp : ");
    scanf("%i", &y);

    /* exercice a: un retangle */
    for (i=1; i<=x; i++)
    {
        for (j=1; j<=y; j++)
        {
            putchar('x');
        }
        putchar('\n');
    }
    putchar('\n');


    /* exercuce c: un triangle */
    for (i=1; i<=x; i++) {
        for (j=1; j<=i; j++) {
            putchar ('x');
        }
        putchar ('\n');
    }
    putchar ('\n');

    /* exercice c: un triangle centralise */
    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');
    }
}
```

Informatique S1 – Programmation C
Exercices – Corrigés

TD 6 : Les tests *if*

Dans ce TD, nous allons réaliser des exercices couvrant l'usage des tests *if* et l'usage des tests emboités dans le langage C.

Objectifs : présentation des tests *if*, des tests emboités, et des opérateurs logiques &&, || et !.

- Présentation test if
- Présentation tests emboités
- Opérateurs logiques &&, || et !

Exercice 1

a) Remplir les trous du code suivant. Que fait ce programme ?

```
#include <stdio.h>

/*  TD 6 : lecture de 3 nb entiers, identification
 *         du plus petit et du plus grand  */

int main () {
   int a , b;

   /* lecture des variables */
   printf ("Entrer a : ");
   scanf ("%d", &a);

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

   if ( a<b ) {
      printf (" a < b \n");
   }
   else {
      printf (" a >= b \n");
   }
}
```

Les étudiants doivent observer que les variables utilisées dans le code sont **a** et **b**.

A travers le message affiché à l'utilisateur, les étudiants peuvent déduire le test **(a<b)** dans le if.

Le complément d'un if est un **else**. ☺

b) Si l'utilisateur entre 3 et 3, il sera affiché le message « a >= b ».

c) Programme qui affiche aussi le message « a ==b » :

```
#include <stdio.h>

/*  TD 6 : lecture de 3 nb entiers, identification
 *         du plus petit et du plus grand  */

int main () {
   int a,b;

   /* lecture des variables */
   printf ("Entrer a : ");
   scanf ("%d", &a);
```

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

   if ( a<b ) {
      printf (" a < b \n");
   }
   else if ( a == b) {
       printf (" a == b \n");
   }
   else {
      printf (" a >= b \n");
   }
}
```

Exercice 2

Programme qui indique le plus petit numéro par 3 n° entiers fourni par l'utilisateur :

```
#include <stdio.h>

/*  TD 6 : lecture de 3 nb entiers, identification
 *         du plus petit */

int main () {
   int a,b,c;
   int petit;

   /* lecture des variables */
   printf ("Entrer a : ");
   scanf ("%d", &a);

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

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

   /* solution simple : on test toutes les combi */
   if ( a<b && a<c ) {
      petit = a;
   }
   else if ( b<a && b<c ) {
       petit = b;
   }
   else if (c<a && c<b) {
      petit = c;
   }
   else //ils sont egaux
      petit = a;

   printf ("Le nombre le plus petit est %d \n", petit);
}
```

Exercice 3

Vérifier si b divise a :

```c
#include <stdio.h>

/* TD 6 : a divise b ? */

int main () {
  int a, b, reste;

  /* lecture des variables */
   printf ("Entrez a : ");
   scanf ("%d", &a);
   printf ("Entrez b : ");
   scanf ("%d", &b);

   /* calcul */
   reste = a % b;
   if (reste == 0) /* ou if (!reste) */
       printf ("b divise a : a%%b = %d \n", reste);
   else
       printf ("b ne divise pas a : a%%b = %d \n", reste);
}
```

Exercice 4

```c
#include <stdio.h>

/* TD 6 : a pair ou impair ? */

int main () {
  int a, reste;

  /* lecture des variables */
   printf ("Entrez a : ");
   scanf ("%d", &a);

   /* calcul */
   reste = a % 2;
   if (reste == 0) /* ou if (!reste) */
       printf ("%d est pair \n", a);
   else
       printf ("%d est impair \n", a);
}
```

Exercice 5

Ecrire un programme qui lit une note *n* (sur 20), puis affiche la mention correspondante : « ajourné » si $n < 10$, « passable » si $10 \leq n < 12$, « assez bien » si $12 \leq n < 14$, « bien » si $14 \leq n < 16$, et « très bien » sinon.

```c
#include <stdio.h>

/*  TD 6: mention d'une note  */

int main () {

   float note;

   printf ("Entrez la note : ");
```

```
   scanf ("%f", &note);

   /* solution 1 : operateurs logiques */
   if (note < 10)
      printf ("Mention Ajourne \n");
   if (note >= 10  && note < 12)
      printf ("Mention Passable \n");
   if (note >= 12 && note < 14)
      printf ("Mention Assez Bien \n");
   if (note >= 14 && note < 16)
      printf ("Mention Bien \n");
   if (note >= 16)
      printf ("Mention Tres Bien \n");

   /* solution 2 : if embriques */
   if (note < 10)
      printf ("Mention Ajourne \n");
   else if (note < 12)
      printf ("Mention Passable \n");
   else if (note < 14)
      printf ("Mention Assez Bien \n");
   else if (note < 16)
      printf ("Mention Bien \n");
   else
      printf ("Mention Tres Bien \n");
}
```

Exercice 6 - Avancé

Etant donné un numéro entier *n* fourni par l'utilisateur, afficher tous les diviseurs de *n*.

```
#include <stdio.h>

/*  TD n : afficher tous les diviseurs de n */

int main () {
   int n, i;

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

   for (i=n; i>0; i--) {
       if ( (n%i) == 0) { /* ou if (!(n%i)) { */
           printf (" %d ",i);
       }
   }
   printf ("\n");
}
```

Exercice 7 - Avancé

a) Calculer la moyenne d'un ensemble de notes fournies par l'utilisateur. Pour terminer le programme, l'utilisateur doit fournir un négatif qui n'est pas pris en compte dans le calcul de la moyenne.

b) Le programme précédent doit afficher également la mention associée à la moyenne.

```c
#include <stdio.h>

/*  TD 6 : calculer la moyenne des notes et afficher la mention */

int main () {
   float note, moy;
   int n = 0;
   moy = 0.0;

   do {
      printf ("Entrez une note (-1 pour terminer) : ");
      scanf ("%f", &note);
      printf ("%f ",note);
      if (note >= 0) {
          moy += note;
          n++;
      }
   } while ( note >= 0);

   moy /= n;

   printf ("%f ",moy);

   if (moy < 10)
      printf ("%f (Mention : insatisfaisante ) \n", moy);
   else if (moy < 12)
      printf ("%f (Mention : Passable) \n", moy);
   else if (moy < 14)
      printf ("%f (Mention : Assez Bien) \n", moy);
   else if (moy < 16)
      printf ("%f (Mention : Bien) \n", moy);
   else
      printf ("%f (Mention : Tres Bien) \n", moy);
}
```

Informatique S1 – Programmation C
Exercices – Corrigés

TD 7 : *switch*

Dans ce TD, nous allons réaliser des exercices couvrant l'usage de l'instruction switch et de l'opérateur « ? : » dans le langage C.

Objectifs : présentation des tests « switch »

- Présentation des tests switch
  - Usage du break
- Opérateur de test ? :

Exercice 1

```
#include <stdio.h>

/*  TD 7 : choix de menu */

int main () {
    int choix;

    /* affichage du menu */
    printf ("Indiquer votre preference :\n");
    printf ("I============================I\n");
    printf ("I [1] - Menu salade           I\n");
    printf ("I [2] - Menu sand. + salade   I\n");
    printf ("I [3] - Menu burger           I\n");
    printf ("I [4] - Menu burger frites    I\n");
    printf ("I [5] - Aucun des precedents  I\n");
    printf ("I============================I\n");
    printf ("\n");

    /* demande le choix */
    printf ("Choix : ");
    scanf ("%d",&choix);

    switch (choix) {
        case 1:
                printf ("Choix : Menu salade \n");
                printf ("        Bravo ! Votre sante vous remercie.\n");
                break;
        case 2:
                printf ("Choix : Menu sans + salade \n");
                printf ("           C'est bien de faire attention a votre
ligne.\n");
                break;
         case 3:
                printf ("Choix : Menu burger \n");
                printf ("        Attention aux repas non-equilibres\n");
                break;
         case 4:
                printf ("Choix : Menu burger frites \n");
                printf ("        Attention a votre cholesterol.\n");
```

```
                    break;
            case 5:
                    printf ("Choix : Aucun des precedents \n");
                    printf  ("               N'oubliez  pas  qu'il  faut  manger
equilibre.\n");
                    break;
            default:
                    printf ("Choix invalide! \n");
    }
}
```

Exercice 2

Note : les étudiant n'ont pas vu en cours les fonctions « `toupper` » et « `tolower` » (bibliothèque `ctype.h`), ni les fonctions de type « `isdigit` », « `isalpha` », etc. (aussi dans la bibliothèque `ctype.h`). Les premières peuvent être expliquées lors du TD, s'il y a l'intérêt de la part des étudiants. Les dernières peuvent être également expliquées, mais l'exercice demande de manière explicite l'usage de l'instruction `switch`.

```
#include <stdio.h>
/* TD 7 : switch voyelle/chiffre/consonne */
int main () {
   char c;

   printf ("Entrez un caractere : ");
   scanf ("%c", &c);

   switch (c) {
      case 'a':  case 'A':
      case 'e':  case 'E':
      case 'i':  case 'I':
      case 'o':  case 'O':
      case 'u':  case 'U':
                 printf ("voyelle \n");
                 break;
      case '0':
      case '1':
      case '2':
      case '3':
      case '4':
      case '5':
      case '6':
      case '7':
      case '8':
      case '9':
                 printf ("chiffre \n");
                 break;
      case 'b': case 'B':
      case 'c': case 'C':
      case 'd': case 'D':
      case 'f': case 'F':
      case 'g': case 'G':
      case 'h': case 'H':
      case 'j': case 'J':
      case 'k': case 'K':
      case 'l': case 'L':
      case 'm': case 'M':
      case 'n': case 'N':
```

```
      case 'p': case 'P':
      case 'q': case 'Q':
      case 'r': case 'R':
      case 's': case 'S':
      case 't': case 'T':
      case 'v': case 'V':
      case 'x': case 'X':
      case 'z': case 'Z':
                printf ("consonne \n");
                break;
      default:
                printf ("%c est unconnu (caractere de controle?) \n",c );
   }
}
```

Exercice 3 :

```
#include <stdio.h>

/*  TD 7 : calculette    */

int main () {
   float a, b, resultat;
   char op;

   //entre des donnees
   printf ("Entrez l'operation [+ - * /]: ");
   op = getchar();
   printf ("Entrez a : ");
   scanf ("%f", &a);
   printf ("Entrez b : ");
   scanf ("%f", &b);

   resultat = 0.0;
   //calculette
   switch (op) {
       case '+':
                 resultat = a + b;
                 break;
       case '-':
                 resultat = a - b;
                 break;
       case '*':
                 resultat = a * b;
                 break;
       case '/':
                 resultat = a / b;
                 break;
      default :
                 printf ("Operateur %c inconnu !\n", op);
   }

   printf ("%f %c %f = %f \n", a, op, b, resultat);

}
```

Exercice 4

```
#include <stdio.h>

/*  TD 7 : operateur ? :    */

int main () {
   int a, b;

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

   printf ("a divise b : %c \n",
                        (b%a == 0) ? 'O' : 'N');
   printf ("b divise a : %c \n",
                        (a%b  == 0) ? 'O' : 'N');
   printf ("a est impair : %c \n",
                        (a%2 != 0) ? 'O' : 'N');
   printf ("b est impair : %c \n",
                        (b%2 != 0) ? 'O' : 'N');
   printf ("a < b : %c \n",
                        (a<b) ? 'O' : 'N');
   printf ("a > b : %c \n",
                        (a>b) ? 'O' : 'N');
}
```

Exercice 5

En cours, ils ont eu un exemple de devinette qui calculait un numéro aléatoire que l'utilisateur devrait trouver. Cet exercice est donc une extension de l'exemple vu en cours, ce qui n'enlève en rien sa complexité. Ils auront probablement des difficultés pour terminer la boucle avant le troisième essai, si l'utilisateur trouve la bonne réponse avant celui-ci (éviter d'utiliser break pour cela), et aussi pour pouvoir montrer la réponse uniquement si l'utilisateur ne l'a pas trouvé.

Note : les étudiant n'ont pas vu « `define` » en cours (donc, pas de `#define FALSE 0`).

```
#include <stdio.h>
#include <stdlib.h>    //fonction rand
#include <time.h>      //fonction time

/* cm7 : swith (devinette) */
int main ()
{
    int essai,x;
    int i;
    int trouve = 0;

    //numero mystere
    srand (time(NULL)); //initialise le generateur
    x = rand()%11;

    //3 essais ou jusqu'a qu'on trouve la reponse
    for (essai=1; essai<=3 && !trouve; essai++) {
        printf ("Essai %d: \nEntrez i [0-10] : ", essai);
        scanf ("%d", &i);

        switch (x-i) {
```

```
            case 0:
                printf ("bingo!!\n");
                trouve = 1; //pour terminer
                break;
            case 1:
            case -1:
                printf ("presque\n");
                break;
            case 2:
            case -2:
            case 3:
            case -3:
                printf ("proche\n");
                break;
            default:
                printf ("loin\n");
        }
    }

    //si on n'a pas trouve la reponse, on la montre
    if (!trouve)
        printf ("Domage! Reponse : %d\n",x);
}
```

Informatique S1 – Programmation C
Exercices – Corrigés

TD 8 : codage binaire et caractères ASCII

Dans ce TD, nous allons réaliser des exercices couvrant la représentation des caractères en langage C.

Note :

- Avant de réaliser les exercices ci-dessous, changer dans le terminal le codage des caractères vers « Occidental » (menu Terminal → Définir le codage des caractères → Occidental (ISO-8859-15).
- Aucun exercice sur la conversion binaire ↔ décimal n'est proposé dans la TD. Si vous pensez qu'il est important d'ajouter ce type d'exercice, n'hésitez pas à le faire.

Exercice 1

Pas de difficulté particulière, il faut juste convertir le caractère lu en numéro entier.

```
#include <stdio.h>

/*  TD 8 : char-ascii*/

int main () {
   char c;
   int ascii;

   printf ("Entrez un caractere : ");
   c = getchar();
   ascii = (int) c;

   printf ("Code ASCII de %c est %d\n",c,ascii);

}
```

Exercice 2

a) Solution la plus simple : une boucle for avec un printf.

```
#include <stdio.h>

/*  TD 8 : afficher ASCII*/

int main () {
  int i;
  int j;

  for (i=0; i<256; i++) {
     printf ("%d = %c \n", i,i);
  }
}
```

b) Exercice qui rappelle certains des concepts vu lors de la TD 5. Pour afficher dans un tableau de 10 colonnes, il faut afficher le caractère '|' après chaque caractère ASCII afin de séparer les colonnes, et afin de séparer les lignes du tableau, il faut compter les colonnes (si colonne divisible par 10, on imprimer le caractère de nouvelle ligne '\n').

Note : certain caractères, surtout les caractères de contrôle, ne vont pas s'afficher correctement, et les premières lignes du tableau ne seront pas forcément alignées aux lignes postérieures (la séparation des colonnes avec le '|' ne sera pas forcément alignée).

```
#include <stdio.h>

/*  TD 8 : char-ascii */

int main () {
  int i;

  /* afficher l'ensemble des caracters ASCII
   * dans un tableau de 10 colonnes */
  putchar ('|');
  for (i=0; i<256; i++) {
      printf (" %c |", i);
      if (i%10 == 0) {
          putchar ('\n');
          putchar ('|');
      }
  }
}
```

Exercice 3

a) Exercice dérivé de l'exercice 8 de la TD 5 (afficher un arbre de x lignes de hauteur). La seule différence réside sur le choix du caractère, le caractère 174 (un « sur windows, ou un ® sous linux). Attention à bien changer le codage comme mentionner o début de la TD.

```
#include <stdio.h>

/*  TD 8 : Arbre de Noel    */

int main () {
    int i, j;
    int hauteur=10;
    char c = (char) 174;

    /* exercice c: un triangle centralise */
    for (i=1; i<=hauteur; i++)
    {
        for (j=1; j<=hauteur-i; j++) {
            putchar(' ');
        }
        for (j=1; j<=2*i-1; j++) {
            putchar(c);
        }
        putchar('\n');
    }
}
```

b) Similaire à l'exercice précédent, mais il faut ajouter un « pied » de 4 lignes dans l'arbre. Pour cela, une boucle supplémentaire est nécessaire, une fois que le reste de l'arbre à été affiché.

```
#include <stdio.h>

/*  TD 8 : Arbre de Noel    */

int main () {
    int i, j;
    int hauteur=10;
    int pied=4;
    char c = (char) 174;

    /* exercice c: un triangle centralise */
    for (i=1; i<=hauteur; i++)
    {
        for (j=1; j<=hauteur-i; j++) {
            putchar(' ');
        }
        for (j=1; j<=2*i-1; j++) {
            putchar(c);
        }
        putchar('\n');
    }

    for (i=1; i<=pied; i++) {
        for (j=1; j<=(hauteur+pied)/2; j++) {
            putchar (' ');
        }

        for (j=1;j<=pied; j++) {
            putchar(c);
        }
        putchar ('\n');
    }
}
```

Exercice 4

a) Cet exercice introduit une complexité supplémentaire : les étudiants, à chaque lecture du clavier, doivent lire 2 caractères, celui fourni par l'utilisateur et le caractère de nouvelle ligne qui reste enregistré dans le buffer de lecture.

```
#include <stdio.h>

/*  TD 8 : char-ascii */

int main () {
   char c;
   char op, nl;

   op = 'O';

   do {
       printf ("Entrez un caractere : ");
       scanf ("%c%c", &c, &nl); //on capture le charactere et le nl
       printf ("Code ASCII de %c est %d\n",c,c);

       printf ("Continuer [ O | N ] ? ");
```

```
        scanf ("%c%c", &op, &nl); //on capture le charactere et le nl
    } while (op=='o' || op=='O');
}
```

b) A partir de l'exercice précédent, ils doivent identifier et compter les majuscules, minuscules et chiffres. Pour cela, ils peuvent soit utiliser soit l'exercice de *switch* fait dans la TD précédente, soit utiliser les caractéristiques propres au code ASCII (les majuscules sont codées entre 65 et 90, les minuscules entre 97 et 122, les chiffres entre 48 et 57).

```
#include <stdio.h>

/*  TD 8 : char-ascii */

int main () {
   char c;
   char op, nl;
   int maj, min, chiffre;
   int ascii;

   op = 'O';
   maj = 0;
   min = 0;
   chiffre = 0;

   do {
       printf ("Entrez un caractere : ");
       scanf ("%c%c", &c, &nl); //on capture le charactere et le nl
       ascii = (int) c;

       printf ("Code ASCII de %c est %d\n",c,ascii);

       //tous les chiffres sont codes entre 48 et 57 en ASCII
       //tous les maj sont codes entre 65 et 90
       //tous les min sont codes entre 97 et 122
       if (ascii>=48 && ascii<=57) chiffre++;
       else if (ascii>=65 && ascii<=90) maj++;
       else if (ascii>=97 && ascii<=122) min++;

       printf ("Continuer [ O | N ] ? ");
       scanf ("%c%c", &op, &nl); //on capture le charactere et le nl
   } while (op=='o' || op=='O');

   printf ("Vous avez entrer : %d chiffres, \n\t %d lettres majuscules"
                              " \n\t %d lettres minuscules\n", chiffre, maj,
min);

}
```

Informatique S1 – Programmation C
Exercices – Corrigés

TD 9-10 : fonctions et arrays

Dans ce TD, nous allons réaliser des exercices couvrant l'usage des fonctions appartenant aux bibliothèques standards et l'usage des arrays unidimensionnels en langage C.

Objectifs :

- Insister sur l'usage de #define
- Insister sur l'usage des fonctions, surtout sur la question 1.

Exercice 1

Attention : à chaque lecture du clavier, deux caractères sont disponibles : celui fourni par l'utilisateur et la nouvelle ligne (touche entrée). Il faut consommer la nouvelle ligne qui reste sur le buffer afin de lire correctement le caractère suivant.

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

#define MAX 10

/*  TD 9-10 : arrays & fonctions */

int main () {
    char entree[MAX];
    int chiffres, maj, min, blanc;
    int i;

    chiffres=0;
    maj=0;
    min=0;
    blanc=0;

    /* entree des donnees */
    for (i=0; i<MAX; i++) {
        printf ("Entrez un caractere : ");
        entree[i] = getchar();
        getchar(); //on consomme la nouvelle ligne

        /* on compte les chiffres, les lettres majuscules
         *  et minuscules, et les espaces en blanc */
        if (isdigit(entree[i]))
            chiffres++;
        if (isalpha(entree[i])) {
            if (isupper(entree[i]))
                maj++;
            if (islower(entree[i]))
                min++;
        }
```

```
        if (isspace(entree[i]))
            blanc++;
    }

    /* afficher le comptage */
    printf ("Chiffres = %d \t espaces = %d \n"
            "Maj. = %d \t Min. = %d \n", chiffres, blanc, maj, min);

    /* afficher l'entre tte en majuscule */
    printf ("Entree :\n");
    for (i=0; i<MAX; i++) {
        putchar (toupper(entree[i]));
    }
    putchar ('\n');
}
```

Exercice 2

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

#define MAX 5

/* TD 9-10 : moyenne   */

int main () {
    float somme, moy;
    float notes[MAX];
    int i;
    float petit, grand;

    petit = FLT_MAX;
    grand = FLT_MIN;

    for (i=0; i<MAX; i++) {
        printf ("Entrez note %d : ",i);
        scanf ("%f", &notes[i]);

        somme += notes[i];

        petit = (petit > notes[i]) ? notes[i] : petit;
        grand = (grand < notes[i]) ? notes[i] : grand;
    }

    moy = somme/MAX;

    printf ("Moyenne = %f \nExtremites = %.4f , %.4f \nEcart = %.4f , %.4f
\n",
             moy, petit, grand, (moy-petit), (moy-grand));

    for (i=0; i<MAX; i++) {
        printf ("Ecart %d : %f\n", i, (moy-notes[i]));
    }
}
```

Exercice 3

```
#include <stdio.h>

#define MAX  5

/*  TD 9-10 : frequence  */

int main () {
   float array[MAX];
   int freq[MAX];
   int i,j, count;

   for (i=0; i<MAX; i++) {
       printf ("Entrez un numero [%d] : ",i);
       scanf ("%f", &array[i]);
   }

   for (i=0; i<MAX; i++) {
       count=0;
       for (j=0; j<MAX; j++) {
           if (array[i] == array[j]) {
               count++;
           }
       }
       freq[i] = count;
   }

   printf ("N. \t Valeur \t Freq \n");

   for (i=0; i<MAX; i++) {
       printf (" %d \t %f \t %d \n",i, array[i], freq[i]);
   }
}
```