INF203 - Exercices semaine 5

C : types, tests, boucles, parcours de tableaux

Exercice 1 :

Soit le programme suivant :

```
#include <stdio.h>
int main() {
    double a=0.5, b=0.25, c=0.1;
    printf("a = %f\n", a);
    printf("b = %f\n", b);
    printf("c = %f\n", c);
    printf("a+b = %f\n", a+b);
    printf("b+c = %f\n", b+c);
    double epsilon = 1 - (a+c+c+c+c+c);
    printf("epsilon = %.20f", epsilon);
    if (epsilon == 0)
        printf (" vaut 0\n");
    else
        printf(" ne vaut pas 0\n");
    return 0;
}
```

1. Qu'affiche-t-il ? pourquoi ?
2. Comment pouvez vous faire pour que l'affichage soit plus pertinent ?
3. Comment pouvez vous faire pour que les calculs soient exacts ?

Solution :

1. Le script retourne "`epsilon = 0.000000 ne vaut pas 0`". N'oublions en effet pas que les doubles sont encodés sous la forme de sous multiples de 2 et que 0.1 n'est qu'approximé à un bit de précision machine près.
2. Un affichage plus pertinent consisterait à comparer la valeur absolue de `epsilon` à un seuil faible restant assez supérieur à la précision machine (par exemple $1e-10$).
3. Pour assurer des calculs exacts, on peut passer par une représentation entière ; par exemple en représentant 0.1 sous la forme du couple d'entiers $(1, 1)$ dans l'écriture $1e-1$. On effectue alors de l'arthmétique exacte sur les entiers.

Exercice 2 :

Étant donnée la fonction suivante :

```
#include <stdlib.h>
#include <unistd.h>

// Genere un entier entre 0 et borne-1
long generer_entier(long borne) {
    static int seme = 0;
    if (!seme) {
        srandom(getpid());
        seme = 1;
    }
    return random() % borne;
}
```

on peut écrire le programme suivant :

```
#include <stdio.h>
#include "generer_entier.c"

int main() {
    long x = generer_entier(100);
    printf("J'ai genere l'entier %ld\n", x);
    return 0;
}
```

1. Écrivez un programme qui génère deux entiers, les affiche et affiche le maximum des deux en utilisant une fonction `max2` que vous écrirez pour l'occasion.
2. Écrivez un programme qui génère trois entiers, les affiche et affiche le maximum des trois
   (a) d'abord en utilisant `max2` ;
   (b) ensuite sans l'utiliser.
3. Même question avec n entiers, où n est un entier défini au début du programme.

Solution :

1.
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

long generer_entier(long borne) {
    static int seme = 0;
    if (!seme) {
        srandom(getpid());
        seme=1;
    }
    return random() % borne;
}

long max2(long x,long y) {
    if (x>y)
        return x;
    else
        return y;
}

int main () {
    long x = generer_entier (100);
    long y = generer_entier (100);
    printf("%ld\n",max2(x,y));
    return 0;
}
```

2. (a)
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

long generer_entier(long borne) {
    static int seme = 0;
    if (!seme) {
        srandom(getpid());
        seme=1;
    }
    return random() % borne;
}
```

```
long max2(long x,long y) {
    if (x>y)
        return x;
    else
        return y;
}

int main () {
    long x = generer_entier (100);
    long y = generer_entier (100);
    long z = generer_entier (100);
    printf("%ld\n",max2(max2(x,y),z));
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

long generer_entier(long borne) {
    static int seme = 0;
    if (!seme) {
        srandom(getpid());
        seme=1;
    }
    return random() % borne;
}

int main () {
    long x = generer_entier (100);
    long y = generer_entier (100);
    long z = generer_entier (100);
    long max = x;
    if (y>max)
        max = y;
    if (z>max)
        max = z;
    printf("%ld\n",max);
    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

long generer_entier(long borne) {
    static int seme = 0;
    if (!seme) {
        srandom(getpid());
        seme=1;
    }
    return random() % borne;
}

long max2(long x,long y) {
    if (x>y)
        return x;
    else
```

```
        return y;
    }

    int main (int argc,char *argv[]) {
        int n = 0;
        sscanf(argv[1],"%d",&n);

        long max = 0;
        for (int i=1;i<=n;i++)
            max = max2(max,generer_entier(100));
        printf("%ld\n",max);
        return 0;
    }
```

Solution :

Pratique de la syntaxe, de printf. Insister sur le fait que les fonctions max2 et max3 calculent et renvoient un résultat, mais N'AFFICHENT RIEN. C'est une confusion qu'on traine trop longtemps.

Exercice 3 :

(13) Répétez la génération d'un nouvel entier jusqu'à ce que celui-ci soit égal à 42.

2. Répétez l'exercice précédent 100 fois (pas à la main !) et calculez le nombre moyen de tirages qu'il a fallu faire pour atteindre 42.

Solution :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

long generer_entier(long borne) {
    static int seme = 0;
    if (!seme) {
        srandom(getpid());
        seme=1;
    }
    return random() % borne;
}

int main () {
    float tot = 0;
    for (int i=0;i<100;i++){
        int count=0;
        while (generer_entier(100) !=42)
            count++;
        tot+=count;
    }
    printf("%f\n",tot/100);
    return 0;
}
```

Exercice 4 :

Écrire un programme en C qui définit un entier `n` et qui affiche la forme suivante, constituée de `n` lignes (ci-dessous exemple avec `n = 5`). Pour le a) par exemple, on affiche `n` lignes, dans chacune desquelles on compte de `1` à `n`.

```
  a. 12345   b.  1        c.       1   d.  12345   e.  12345   f.       1
     12345        12              12        1234          1234          121
     12345        123            123        123            123         12321
     12345        1234          1234        12              12        1234321
     12345        12345        12345        1                1       123454321
```

Solution :

```
#include <stdio.h>

int main(){
    int n=5;
    char cas = 'f';

    switch (cas){
        case 'a':
            {
            char number[n];
            for (int i=0;i<n;i++)
                number[i]= i+49;

            for (int i=0;i<n;i++)
                printf("%s\n",number);
            }
            break;

        case 'b':
            {
            for (int i=1;i<=n;i++){
                char number[n];
                for (int j=0;j<i;j++)
                    number[j]=j+49;
                printf("%s\n",number);
            }
            }
            break;

        case 'c':
            {
            for (int i=1;i<=n;i++){
                char number[n];
                for (int j=0;j<n;j++)
                    number[j]=' ';

                for (int j=0;j<i;j++)
                    number[n-i+j]=j+49;
                printf("%s\n",number);
            }
            }

        case 'f':
            {
            for (int i=1;i<=n;i++){
                char number[n*2-1];
                for (int j=0;j<n*2-1;j++)
                    number[j]=' ';

                for (int j=0;j<i;j++){
                    number[n-i+j]=j+49;
                    number[n-2+i-j]=j+49;
```

```
                }
                printf("%s\n",number);
            }
            }
            break;
    }
            return 0;
}
```

Exercice 5 :

1. Utilisez le générateur fourni pour remplir un tableau de 10 entiers avec des valeurs aléatoires comprises entre 0 et 99 et l'afficher.
2. Écrivez une fonction pour calculer la moyenne des éléments d'un tableau et complétez le programme précédent pour afficher la moyenne des éléments d'un tableau généré.
3. Écrivez une fonction permettant de chercher un élément de valeur donnée dans un tableau donné. Utilisez-la dans le programme précédent pour indiquer si le tableau contient 0, 42 ou 99.

Solution :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

float moy(int *list){
    float tmp = 0;
    for (int i=0;i<10;i++)
        tmp+=list[i];
    return tmp/10;
}

int find(int element,int *list){
    int i=0;
        while (i<10){
        if (list[i]==element)
            return 1;
        i++;
    }
    return 0;
}

int main(){
    srandom(getpid());
    int list[10];
    for (int i=0;i<10;i++)
        list[i] = random() % 100;
    printf("Moyenne: %1.2f\n",moy(list));
    printf("Trouvé 12?: %d\n",find(12,list));
}
```

Exercice 6 :

On peut déterminer si un entier n est un nombre premier en déterminant s'il est divisible par l'un des nombres premiers compris dans l'intervalle $[0; n-1]$. On peut énumérer les nombres premiers compris entre 2 et N de la manière suivante :

— on considère une liste L, initialement vide

— pour chaque entier n compris entre 2 et N, dans l'ordre :
  — si n est divisible par l'un des entiers de L alors il n'est pas premier
  — sinon, ajouter n à L

Cet algorithme s'appelle le crible d'Érastosthène. La liste L peut être stockée dans un tableau de taille suffisante en conservant sa taille dans une variable entière. On peut tester si un nombre x divise un nombre n avec l'opérateur `%` (modulo) : `n%x` est le reste de la division entière de n par x ; s'il vaut 0, x divise n. En utilisant cet algorithme, énumérez les nombres premiers compris entre 2 et 100.

Solution :

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
    int L[100]={2};
    int taille=1;
    for (int i=2;i<100;i++){
        int j=0;
        for (j=0;j<taille;j++){
            if (i % L[j] == 0)
                break;
        }
        if (j==taille)
            L[taille++]=i;
    }

    for (int i=0;i<taille;i++)
        printf("%d\t",L[i]);
    printf("\n");

    return 0;
}
```

Exercice 7 :

Qu'affiche le programme suivant ?

```
#include <stdio.h>
int main() {
    int qd=0b101010;
    printf("En base 8 : %o, en base 10 : %d, en base 16 : %x\n", qd, qd, qd);
    return 0; }
```

Solution :

Le programme retourne l'expression de la valeur `qd` définie en base 2 (et égale à 42 en base décimale) dans plusieurs bases.