INF203 - Exercices semaine 8
C : compléments, révisions

Exercice 1 :

On rappelle comment générer des nombres entiers aléatoires :

```
#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;
}
```

1. Comment pouvez-vous utiliser ce générateur pour générer un nombre à virgule flottante dans $[0; 1[$?
2. Utilisez ce nouveau générateur pour générer le tirage d'un dé à 6 faces pipé ; vous devez générer un entier entre 1 et 6 tel que le dé a une probabilité de :
   — 4/13 d'atterrir sur la face 6 ;
   — 2/13 d'atterrir sur les faces 2, 3, 4 ou 5 ;
   — 1/13 d'atterrir sur la face 1.

Solution :

1. on peut transformer l'entier en `float` et diviser (division en `float`) par l'entier maximal

   ```
   float val=(float)(generer_entier(RAND_MAX))/(float)(RAND_MAX);
   ```

2. on obtient le programme suivant

   ```
   #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 dice(){
       float val=(float)(generer_entier(RAND_MAX))/(float)(RAND_MAX);

       if (val<4./13.)
           return 6;
   ```

```
    if (val<6./13.)
        return 2;
    if (val<8./13.)
        return 3;
    if (val<10./13.)
        return 4;
    if (val<12./13.)
        return 5;
    return 1;
};

int main(){
    int tirages=1300;
    int resultats[6]={0,0,0,0,0,0};
    for (int i=0;i<tirages;i++){
        int tirage=dice();
        printf("Tirage du dé %2d:\t %d\n",i,tirage);
        resultats[tirage-1]+=1;
    }
    printf("\n");
    for (int i=0;i<6;i++)
        printf("Nombre de %d: %d\n",i+1,resultats[i]);
    return 0;
}
```

Exercice 2 :

On considère dans cet exercice deux tableaux d'entiers `A` et `B`, de tailles respectives nA et nB, dont on suppose les valeurs triées par ordre croissant. Les valeurs dans `A` et `B` sont toutes distinctes.

1. Écrivez une fonction permettant de déterminer si `A` est inclus dans `B` (si toutes les valeurs de `A` sont également présentes dans `B`).
2. Écrivez une fonction permettant de remplir en ordre croissant un tableau `C` (de taille $nA + nB$) avec toutes les valeurs de `A` et de `B`.

Solution :

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

int inclus(int *A,int nA,int *B,int nB){
    int i=0;
    int j=0;
    while(i<nA){
        if(j==nB)
            return 0;

        printf("Comparing %d to %d\n",A[i],B[j]);
        if(A[i]>B[j]){
            j++;
        }
        else if(A[i]==B[j]){
            i++;
            j++;
        }
        else
            return 0;
    }
    return 1;
};
```

```
int main(){
    int A[7]={1,4,7,9,13,14,18};
    int B[10]={1,3,4,7,7,10,13,14,17,18};

    if (inclus(A,7,B,10))
        printf("Inclus!\n");
    else
        printf("Non inclus!\n");

    return 0;
}
```

2.
```
#include<stdio.h>

void concatenate(int *A,int nA,int *B,int nB,int *C){
    int i=0;
    int j=0;
    while(i<nA && j<nB){
        if(*A<*B){
            *C++ = *A++;
            i++;
        }
        else{
            *C++ = *B++;
                j++;
        }
    }
    if(i==nA)
        while(j<nB){
            *C++ = *B++;
            j++;
        }
    if(j==nB)
        while(i<nA){
            *C++ = *A++;
            i++;
        }
};

int main(){
    int A[7]={1,4,7,9,13,14,15};
    int B[10]={1,3,4,7,7,10,13,14,17,18};

    int C[7+10];
    concatenate(A,7,B,10,C);

    printf("Union classée des deux tableaux: ");
    for(int i=0;i<7+10;i++)
        printf("%d ",C[i]);

    printf("\n");

    return 0;
}
```

Exercice 3 :

1. Écrivez un programme qui lit un fichier texte, et qui mémorise chacun de ses mots (sous la forme d'une chaîne de caractères) dans un tableau (qu'on suppose déclaré suffisamment grand).
   Les mots du texte sont séparés par des espaces ou des retours à la ligne.
2. Améliorez votre programme pour que les mots présents en plusieurs exemplaires dans le fichier ne soient mémorisés qu'une seule fois dans le tableau.
3. Modifiez encore votre programme pour mémoriser également le nombre d'occurrences de chaque mot.
4. Enfin, faites en sorte que le fichier dans lequel on lit les mots soit donné comme un argument en ligne de commande de votre programme.
   Vous effectuerez les vérifications d'usage : que le nombre d'arguments fournis est le bon, que le fichier donné existe et qu'il est accessible en lecture.

Solution :

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


int main(){
    FILE *f=fopen("test","r");

    char mots[100][200];
    int nombre_mots=0;

    while(!feof(f)){
        fscanf(f,"%s",mots[i++]);
        nombre_mots++;
    }

    for(int i=1;i<=nombre_mots;i++)
        printf("%s\t",mots[i]);

    return 0;
}
```

2./3.
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>


int main(){
    FILE *f=fopen("test","r");

    struct mots {
        char mot[200];
        int occurrence;
    };

    struct mots les_mots[100];
    int nombre_mots=0;

    while(!feof(f)){
        char le_mot[200];
        int mot_trouve=0;
```

```
            fscanf(f,"%s",le_mot);
            int i;
            for(i=1;i<=nombre_mots;i++)
                if(strcmp(les_mots[i].mot,le_mot)==0){
                    les_mots[i].occurrence++;
                    mot_trouve = 1;
                }
            if(!mot_trouve){
                printf("Je scanne: \"%s\"\n",le_mot);
                strcpy(les_mots[nombre_mots+1].mot,le_mot);
                les_mots[nombre_mots+1].occurrence=1;
                nombre_mots++;
            }
        }

        for(int i=1;i<=nombre_mots;i++)
            printf("%s [%d]\t",les_mots[i].mot,les_mots[i].occurrence);
        return 0;
    }
```

Exercice 4 :

Voici le contenu de six fichiers écrits en langage `C` présents dans un même répertoire :

hop.c	travail.c

`int main(void) {`
`  char x;`
`  x = choix();`
`  disp(1, x);`
`  return 0;`
`}`
	`char choix() {`
	`  if (decide(0, 1))`
	`  return 'c';`
	`  else`
	`  return 'd';`
	`}`

util.h	**trucs.h**
`void disp(int a, char b);`	`int decide(int a, int b);`
trucs.c	**util.c**

`int decide(int a, int b) {`
`  if (a == 0)`
`  return b;`
`  else`
`  return !b;`
`}`
	`void disp(int a, char b) {`
	`  if (decide(a, a))`
	`  printf("%c", b);`
	`}`

1. Quelles lignes `#include ...` faut-il ajouter dans quels fichiers pour que le programme soit correctement structuré et qu'on puisse le compiler ?
2. Y a-t-il besoin d'un ou plusieurs fichiers supplémentaires ?
   Si oui, écrire leur contenu.

Solution :

1. Il faut ajouter

— à `hop.c` : `#include "util.h"` (parce que `disp` doit être défini) puis `#include "travail.c"` (parce que `choix` doit être défini)
— à `travail.c` : `#include "trucs.h"` pour définir `decide`
— à `trucs.c` : n'a besoin de rien.
— à `util.c` : `#include<stdio.h>` pour que `printf` soit défini, puis `#include "trucs.h"` pour que `decide` soit défini.

On remarque cependant que `trucs.h` est appelé par `travail.c` et par `util.c`. Il sera nécessaire d'encadrer les codes des fichiers `.h` par exemple d'un `#ifndef __TRUCS_H__ #define __TRUCS_H__ #endif`.

Il est aussi une bonne pratique d'inclure `x.h` dans les fichiers `x.c` afin de repérer une erreur de typage au moment de la compilation (plutôt qu'au moment du lien).

2. Il peut être pertinent de créer le fichier `travail.h` qui définit `choix` et de l'inclure à `hop.c` en lieu et place de `travail.c`.