TP 3 - C

Les Tps se font sous un environnement linux, vous pouvez utiliser l'éditeur de texte que vous preferez.

Partie I

1. Sqrt

Dans cet exercice nous allons estimer la racine carrée de nombres. Pour cela, nous utiliserons la méthode de newton qui consite à améliorer une estimation plusieurs fois:

- On commence par une estimation x_0 (par exemple on peut prendre le nombre dont on veut calculer la racine).
- On améliore plusieurs fois l'estimation avec la formule suivante : $x_{n+1} = x_n - \frac{(x_n)^2 - k}{2 * x_n}$ ou x_n est l'estimation courante, x_{n+1} est l'estimation améliorée et k est la valeur dont on cherche à caluler la racine.

1. Ecrivez un programme qui applique cette formule pour estimer la valeur de la racine carrée de cinq (Vérifiez que vous trouver bien 3 à la première iteration puis 2.3 à la seconde itération).
2. Créez une fonction qui prend en argument un flottant qui renvoie l'estimation de sa racine après 15 itérations (verifiez que vous trouvez de bons résultats pour plusieurs nombres, par exemple 4, 5, 9, 57, 573).
3. Sauvegardez cette fonction pour l'exercice suivant.

2. Distance

1. Ecrivez une fonction qui permet de calculer la distance entre deux points en 2 dimensions, elle prendra en arguments les coordonnées de ces points sous la forme de 4 flottants.
2. Ecrivez une fonction qui prend en argument les coordonnées de deux cercles ainsi que leur rayon et qui renvoie 1 si ils ne se touchent pas, 0 sinon.

3. Random :

Dans cet exercice nous allons générer des nombres pseudo-aléatoires. Pour cela nous allons utiliser un générateur congruentiel linéaire.

Pour générer un nombre aléatoire nous allons procéder ainsi:

- Choisir un premier nombre appelé "seed" ou "graine".
- Appliquer une opération sur ce nombre pour en produire un suivant.
- Répéter l'opération à chaque nouveau nombre.

1. Créez un programme qui :
    - demande trois nombres à l'utilisateur: "a", "c" et "m".
    - demande une "graine" à l'utilisateur.
    - affiche le résultat de $(\text{graine} * a + c)\%m$.
2. Modifiez le programme pour qu'il demande un nombre n et affiche n nombres aléatoires.
    - La graine sera remplacée par le nombre généré à chaque nouvelle itération.
    - Pour $a = 25$, $c = 16$, $m = 256$ et graine $= 125$ vous devriez obtenir la suite $69, 205, 21, 29, 229, 109, 181, 189, 133, \ldots$
3. Créez une fonction "random" qui utilise une variable statique pour la graine et qui renvoie la nouvelle valeur à chaque appel.
    - Utilisez les valeurs $a = 75$, $c = 74$, $m = 65537$ et sauvegardez votre code avant de passer à l'exercice suivant

Note (culture générale): Les fonctions des langages de programation java, C et C# (pour ne citer qu'eux) utilisent cet algorithme pour générer des nombres aléatoires.

- *certaines implémentations de C et C++ jusqu'a la version C++11 utilisent les bits 30 à 16 du nombre généré par les paramètres suivants:* $a = 1103515245$, $c = 12345$, $m = 2^{31}$
- *java utilise les 32 bits les plus significatifs du nombre généré avec les paramètres suivants:* $a = 25214903917$, $c = 11$, $m = 2^{48}$

L'implémentation de ces algorithmes peut être consultée (ici pour java: https://developer.classpath.org/doc/java/util/Random-source.html, ici pour C: https://sourceware.org/git/?p=glibc.git;a=blob;f=stdlib/random_r.c)

- *L'implémentation C# n'est pas imposée par le standard et dépend de l'environnement .net installé.*

4. Jeu de dés

1. Modifiez la fonction aléatoire précédente pour utiliser une valeur de départ différente à chaque execution.
   - Pour cela utilisez la fonction suivante :

```c
#include "time.h"
int premiere_valeur(){
    return (int) time(NULL) % 16;
}
```

2. Ci dessous le code d'un casino en ligne qui propose de miser de l'argent sur le résultat d'un jet de dé.

   Recopiez la fonction main ci-dessous ainsi que votre fonction de génération de nombres aléatoires dans un fichier et compilez le.

```c
int demande_mise(int argent);
int demande_pari();

int main(){
    int argent = 50;
    int mise, pari;

    while(argent > 0 && argent < 30000){
        printf("vous avez %d euros\n", argent);
        mise = demande_mise(argent);
        printf("vous misez : %d\n", mise);
        pari = demande_pari();
        printf("vous avez parié : %d\n", pari);
        int resultat = 1 + alea() % 6;
        printf("resultat du lancer : %d\n", resultat);
        if (pari == resultat){
            printf("vous gagnez : %d\n", 2 * mise);
            argent += 2*mise;
        } else {
            printf("vous perdez la mise\n");
            argent -= mise;
        }
    }
    if (argent){
        printf ("bravo, vous avez ruiné le casino\n");
    } else {
        printf("vous êtes ruinés\n");
    }
}
```

   1. Ecrivez le code des deux fonctions demande_mise et demande pari, ces fonctions devront redemander à l'utilisateur de saisir la valeur tant qu'elle n'est pas valide

Partie II : Exercices Avancés

Ces exercices peuvent être réalisés dans l'ordre que vous souhaitez

5. Pas de chance

1. Reprenez le code de l'exercice "jeu de dés" et créez un programme qui vous aidera à ruiner le casino.
2. Essayez en utilisant dans le code les paramètres suivants: $a = 25214903917$, $c = 11$, $m = 2^{48}$.

6. Fonctions variadiques

Dans le cours, nous avons vu qu'une fonction accepte un nombre fixe d'arguments, pourtant ce n'est pas le cas de printf et scanf.

Par exemple `printf("bonjour")` est valide et `printf("%d%d%d", 1,  2, 3)` l'est aussi, bien qu'elles acceptent respectivement 1 et 4 arguments.

On dit que printf et scanf sont variadiques car elles acceptent un nombre variable d'arguments.

Pour utiliser les arguments variadiques, on doit importer `"stdarg.h"` :

Ci dessous un exemple de programe utilisant une fonction variadique.

```c
#include "stdio.h"
#include "stdarg.h"

// la fonction my_sum permet de calculer la somme d'un nombre arbitraire de valeurs
// voir les exemples d'utilisation dans le main
// count contient le nombre d'arguments que nous allons recevoir
int my_sum(int count, ...){
 // on déclare la liste d'arguments passés a la fonction
 va_list liste_args;
 // on initialise la liste en lui passant le nom de l'argument qui la précède
 va_start(liste_args, count);

 int sum = 0;
 for (int i = 0; i < count; i++){
    // on récupère le prochain argument
   sum += va_arg(liste_args, int);
 }
  // on "libere" la liste
 va_end(liste_args);
 return sum;
}


int main(){
  // on appele my_sum en précisant le nombre d'arguments a l'avance
    printf("%d\n", my_sum(3, 1, 2, 3));
    printf("%d\n", my_sum(1, 5));
    printf("%d\n", my_sum(7, 1, 2, 3, 4, 5, 4, 3));
}
```

Explications:

Pour déclarer une fonction variadique, on écrit « ... » en dernier argument de la fonction (par exemple on écrirait `printf(char* format, …)`).

Ensuite on crée une variable de type va_list qui contiendra la liste d'arguments passés à la fonction. On initialise cette variable avec la macro `va_start()`, cette macro accepte en arguments la liste ainsi que le dernier argument passé a la fonction avant cette liste.

Enfin, on peut acceder aux arguments avec la macro `va_arg()`.

va_arg prend la liste d'arguments en premier paramètre et le type attendu en second paramètre, cette macro renvoie le prochain element de la liste et passe au suivant.

1. Reprendre le code précédent et l'adapter pour créer une fonction my_prod qui calcule le produit de tous les arguments passés en paramètres (On aurait ainsi `my_prod(4, 1, 2, 3, 1) = 6`).
2. Ecrire une fonction `my_printf` prenant en paramètres une chaine de format et une liste de charactères et qui affiche la chaine de format en remplacant les ' %' par les charactères dans l'ordre ou ils sont passés. Pour afficher un charactère à l'écran utilisez la fonction `putchar`.

   Exemple :

   ```
   my_printf(« % est une lettre mais pas %\n», 'I', '1') ;
   ```

   affichera à l'écran : `"I est une lettre mais pas 1"`
3. Adapter cette fonction pour qu'on puisse afficher des chaines de charactères en utilisant le format `'$'`