Exercices d'entraînement

Ces exercices sont importants pour ton entraînement personnel. Pour développer au mieux tes capacités de réflexions et de résolutions de problèmes, ne cherche pas des méthodes de résolutions ni les résultats sur le net (çà te desservira). Merci pour ta compréhension.

Echauffement

Etape 1.

Ecrire un algorithme qui permet de lister les 100 premiers nombres premiers. Stocker ensuite ces nombres dans un tableau T. Enregistrer votre code dans un fichier nommé echauff_et1.c

Etape 2.

Etant donné une chaine de caractère cdc = 'je' et étant donné des entiers entre 0 et 100.
Ecrire un algorithme permettant de rajouter à cette chaine de caractère 'bla' si le nombre est multiple de 3, 'tchip' s'il est multiple de 5 et 'netflix' s'il est à la fois multiple de 3 et de 5. Enregistrer votre code dans un fichier echauff_et2.c

Etape 3.

Ecrire un algorithme permettant de calculer la somme et la moyenne des 100 premiers entiers. Enregistrer dans un fichier echauff_et3.c

Ecrire ensuite un algorithme permettant de faire l'échange de valeur entre deux variables entières a et b. Enregistrer dans un fichier echauff_et3_bis.c

Entrons dans le vif du sujet

Exercice 1 - Le tri à bulle

L'objectif de l'exercice est de générer aléatoirement un nombre m d'entier et d'appliquer sur ces nombres le tri à bulle qui est un algorithme de tri.

Créer un fichier exo1.c dans votre éditeur préféré. Dans les en-têtes d'inclusion, inclure la bibliothèque de manipulation du temps

`#include<time.h>`. N'oubliez pas d'inclure aussi la bibliothèque `#include<stdlib.h>`.

Ajouter dans votre fichier dans la zone d'initialisation et hors de la méthode **main** le bout de code suivant : `srand(time(NULL));`. Cette instruction permet d'initialiser la variable génératrice de nombre aléatoire **srand** sur le temps actuel. Pour en savoir plus sur cette fonction, vous pouvez consulter le lien suivant : https://www.tutorialspoint.com/c_standard_library/c_function_srand.htm .

En bas de votre fichier et hors de la fonction principale main, ajouter le code de la fonction suivant :

```
int aleatoire(int a, int b) {
    return rand()%(b-a) + a;
}
```

. Cette fonction prend en argument ou entrée deux entiers a et b et renvoie en sortie un nombre aléatoire compris entre a et b.

Créer ensuite deux fonctions que l'on nommera **maxval** et **minval**. La fonction maxval prendra en argument deux entiers a et b et retournera en sortie la valeur du plus grand des deux. De même la fonction minval, prendra en argument deux entiers a et b puis renverra en sortie le plus petit des deux. Tester les deux fonctions pour s'assurer qu'elles fonctionnent correctement. Vous pouvez maintenant effacer votre code de test.

Créer ensuite la fonction que l'on nommera **permuter**. Cette fonction prendra en argument deux entiers a et b et permutera via la procédure leurs valeurs. Voici la structure de la fonction. Je vous laisse le soin de la compléter.

```
void permuter(int *a, int *b){

}
```

Créer un tableau de m éléments. La valeur de m est demandée à l'utilisateur à l'exécution du programme. Stocker dans un tableau d'entier à l'aide d'une boucle les m variables aléatoires en évitant les doublons. Pour ce faire, utiliser la fonction **aleatoire**.

Le tri à bulle c'est quoi ?

Le **tri à bulles** ou **tri par propagation** est un algorithme de tri. Il consiste à comparer répétitivement les éléments consécutifs d'un tableau, et à les permuter lorsqu'ils sont mal triés. Il doit son nom au fait qu'il déplace

rapidement les plus grands éléments en fin de tableau, comme des bulles d'air qui remonteraient rapidement à la surface d'un liquide.

Principe :

L'algorithme parcourt le tableau et compare les éléments consécutifs. Lorsque deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.

Après un premier parcours complet du tableau, le plus grand élément est forcément en fin de tableau, à sa position définitive. En effet, aussitôt que le plus grand élément est rencontré durant le parcours, il est mal trié par rapport à tous les éléments suivants, donc échangé à chaque fois jusqu'à la fin du parcours.

Après le premier parcours, le plus grand élément étant à sa position définitive, il n'a plus à être traité. Le reste du tableau est en revanche encore en désordre. Il faut donc le parcourir à nouveau, en s'arrêtant à l'avant-dernier élément. Après ce deuxième parcours, les deux plus grands éléments sont à leur position définitive. Il faut donc répéter les parcours du tableau, jusqu'à ce que les deux plus petits éléments soient placés à leur position définitive.

Implémenter cet algorithme pour trier les nombres contenus dans le tableau. Afficher le tableau initial et le tableau trié.

Conseil : Pour comparer deux nombres consécutifs vous pouvez utiliser les fonctions minval, maxval ou vous pouvez décider de faire des comparaisons. Pour échanger les valeurs utiliser la méthode permuter.

Exercice 2 – Devine moi

Un pépé se dit un matin, si je pouvais avoir un jeu de devinette de nombre sur mon ordinateur, je pourrais le jouer toute la journée. Forte de tes talents en programmation, tu lui demande à quoi ressemblerait ce jeu. Voici la description du jeu.

Au démarrage de chaque partie du jeu, j'aimerais que l'ordinateur choisisse au hasard un chiffre compris entre 0 et 100. Il me demandera alors de deviner ce nombre en n tentatives, (n<10) que je choisirai moi-même. A chaque proposition, l'ordinateur devra me donner des indices si le nombre a deviné est plus petit, plus grand ou si j'ai réussi à le trouver. Dans le cas où j'ai trouvé le nombre qu'il m'affiche le nombre de tentatives que j'ai effectuée avant de le trouver. Si je n'arrive pas à trouver le mot en n tentatives, j'aurais échoué et le jeu relancera une nouvelle partie. A la fin de chaque partie, j'aimerais que l'ordinateur me demande si je veux rejouer ou pas. Si oui une autre partie est lancée bien sûr si non que le jeu s'arrête.

Programmer ce jeu et enregistrer le dans un fichier **devinemoi.c**

Exercice 4 – Le pendu

Le pendu est un grand classique de jeu de lettre et nous y jouions tous lorsque nous étions gamins vous rappelez-vous ? Pour aujourd'hui nous allons programmer ce jeu pour nous rappeler les anciens moments.

Avant de vous plonger tête baissée dans ce code, révisez bien vos notions sur les pointeurs, les chaines de caractères, les fichiers, les tableaux, les entrées et sorties écran.

Consignes :

Le but du pendu est de retrouver un mot caché en moins de 10 essais. Vous pouvez changer ce nombre maximal pour corser la difficulté bien sûr ! 😉

Déroulement d'une partie :

Supposons que le mot caché soit **ROUGE**.

Vous proposez une lettre à l'ordinateur, par exemple la lettre A. L'ordinateur vérifie si cette lettre se trouve dans le mot caché.

Rappelez-vous : il y a une fonction toute prête dans **string.h** pour rechercher une lettre dans un mot ! Notez que vous n'êtes cependant pas obligés de l'utiliser (personnellement, je ne m'en suis pas servi).

À partir de là, deux possibilités :

- La lettre se trouve effectivement dans le mot : dans ce cas, on dévoile le mot avec les lettres qu'on a déjà trouvées ;
- La lettre ne se trouve pas dans le mot (c'est le cas ici, car A n'est pas dans ROUGE 😋 on indique au joueur que la lettre ne s'y trouve pas et on diminue le nombre de coups restants. Quand il ne nous reste plus de coups (0 coup), le jeu est terminé et on a perdu.

Dans un « vrai » Pendu, il y aurait normalement le dessin d'un bonhomme qui se fait pendre au fur et à mesure que l'on fait des erreurs. En console, ce serait un peu trop difficile de dessiner un bonhomme qui se fait pendre rien qu'avec du texte, on va donc se contenter d'afficher une simple phrase comme « Il vous reste X coups avant une mort certaine ».

Supposons maintenant que le joueur tape la lettre G. Celle-ci se trouve dans le mot caché, donc on ne diminue pas le nombre de coups restants au joueur. On affiche le mot secret avec les lettres qu'on a déjà découvertes, c'est-à-dire quelque chose comme ça :

Si ensuite on tape un R, comme la lettre s'y trouve, on l'ajoute à la liste des lettres trouvées et on affiche à nouveau le mot avec les lettres déjà découvertes :

```
Mot secret : R**G*
```

Le cas des lettres multiples

Dans certains mots, une même lettre peut apparaître deux ou trois fois, voire plus !
Par exemple, il y a deux Z dans PUZZLE ; de même, il y a trois E dans ELEMENT.

Que fait-on dans un cas comme ça ? Les règles du Pendu sont claires : si le joueur tape la lettre E, toutes les lettres E du mot ELEMENT doivent être découvertes d'un seul coup :

```
Mot secret : E*E*E**
```

Il ne faut donc pas avoir à taper trois fois la lettre E pour que tous les E soient découverts.

Exemple d'une partie complète

Voici à quoi devrait ressembler une partie complète en console lorsque votre programme sera terminé :

```
Bienvenue dans le Pendu !

Il vous reste 10 coups a jouer
Quel est le mot secret ? ******
Proposez une lettre : E

Il vous reste 9 coups a jouer
Quel est le mot secret ? ******
Proposez une lettre : A

Il vous reste 9 coups a jouer
Quel est le mot secret ? *A****
Proposez une lettre : O

Il vous reste 9 coups a jouer
Quel est le mot secret ? *A**O*
Proposez une lettre :
```

Et ainsi de suite jusqu'à ce que le joueur ait découvert toutes les lettres du mot (ou bien qu'il ne lui reste plus de coups à jouer 😂

```
Il vous reste 8 coups a jouer
Quel est le mot secret ? MA**ON
Proposez une lettre : R

Gagne ! Le mot secret etait bien : MARRON
```

Saisie d'une lettre en console

La lecture d'une lettre dans la console est plus compliquée qu'il n'y paraît. Intuitivement, pour récupérer un caractère, vous devriez avoir pensé à :

```
1 scanf("%c", &maLettre);
```

Et effectivement, c'est bien **%c** indique que l'on attend un caractère, qu'on stockera dans **maLettre** (une variable de type **char**).

Tout se passe très bien tant qu'on ne refait pas un **scanf**. En effet, vous pouvez tester le code suivant :

```
1  int main(int argc, char* argv[])
2  {
3      char maLettre = 0;
4
5      scanf("%c", &maLettre);
6      printf("%c", maLettre);
7
8      scanf("%c", &maLettre);
9      printf("%c", maLettre);
10
11     return 0;
12 }
```

Normalement, ce code est censé vous demander une lettre et vous l'afficher, et cela deux fois. Testez.

Que se passe-t-il ?

Vous entrez une lettre, d'accord, mais… le programme s'arrête tout de suite après, il ne vous demande pas la seconde lettre ! On dirait qu'il ignore le second **scanf**.

Que s'est-il passé ?

En fait, quand vous entrez du texte en console, tout ce que vous tapez est stocké quelque part en mémoire, *y compris l'appui sur la touche* ***Entrée*** (**\n**).

Ainsi, la première fois que vous entrez une lettre (par exemple A) puis que vous appuyez sur *Entrée*, c'est la lettre A qui est renvoyée par le **scanf**. Mais la seconde fois, **scanf** renvoie le**\n** correspondant à la touche ***Entrée*** que vous aviez pressée auparavant !

Pour éviter cela, le mieux c'est de créer notre propre petite fonction **lireCaractere () :**

```c
char lireCaractere()
{
    char caractere = 0;

    caractere = getchar(); // On lit le premier caractère
    caractere = toupper(caractere); // On met la lettre en majuscule si elle ne l'est pas déjà

    // On lit les autres caractères mémorisés un à un jusqu'au \n (pour les effacer)
    while (getchar() != '\n') ;

    return caractere; // On retourne le premier caractère qu'on a lu
}
```

Cette fonction utilise **getchar ()** qui est une fonction de **stdio** qui revient exactement à écrire **scanf ("%c", &lettre);** La fonction **getchar** renvoie le caractère que le joueur a tapé.

Cela étant, j'utilise une fonction standard **toupper ()**. Cette fonction transforme la lettre indiquée en majuscule. Comme ça, le jeu fonctionnera même si le joueur tape des lettres minuscules. Il faudra inclure **ctype.h** pour pouvoir utiliser cette fonction (ne l'oubliez pas 😊).

Vient ensuite la partie la plus intéressante : celle où je vide les autres caractères qui auraient pu avoir été tapés. En effet, en rappelant **getchar** on prend le caractère suivant que l'utilisateur a tapé (par exemple l'*Entrée*\n). Ce que je fais est simple et tient en une ligne : j'appelle la fonction **getchar** en boucle jusqu'à tomber sur le caractère **\n**. La boucle s'arrête alors, ce qui signifie qu'on a « lu » tous les autres caractères, ils ont donc été vidés de la mémoire. On dit qu'on **vide le buffer**.

Pourquoi y a-t-il un point-virgule à la fin du **while** et pourquoi ne voit-on pas d'accolades ?

En fait, je fais une boucle qui ne contient pas d'instructions (la seule instruction, c'est le **getchar** entre les parenthèses). Les accolades ne sont pas nécessaires vu que je n'ai rien d'autre à faire qu'un **getchar**. Je mets donc un point-virgule pour remplacer les accolades. Ce point-virgule signifie « ne rien faire à chaque passage dans la boucle ». C'est un peu particulier je le reconnais, mais c'est une technique à connaître, technique qu'utilisent les programmeurs pour faire des boucles très courtes et très simples.

Dites-vous que le **while** aurait aussi pu être écrit comme ceci :

```c
while (getchar() != '\n')
{

}
```

Il n'y a rien entre accolades, c'est volontaire, vu qu'on n'a rien d'autre à faire. Ma technique consistant à placer juste un point-virgule est simplement plus courte que celle des accolades.

Enfin, la fonction **lireCaracterere** tourne le premier caractère qu'elle a lu : la variable **caractere**.

En résumé, pour récupérer une lettre dans votre code, vous n'utiliserez pas :

```
1 scanf("%c", &maLettre);
```

Vous utiliserez à la place notre super-fonction :

```
1 maLettre = lireCaractere();
```

Dictionnaire de mots

Dans un premier temps pour vos tests, je vais vous demander de fixer le mot secret directement dans votre code. Vous écrirez donc par exemple :

```
1 char motSecret[] = "MARRON";
```

Alors oui, bien sûr, le mot secret sera toujours le même si on laisse ça comme ça, ce qui n'est pas très rigolo. Je vous demande de faire comme ça dans un premier temps pour ne pas mélanger les problèmes. En effet, une fois que votre jeu de Pendu fonctionnera correctement (et seulement à partir de ce moment-là), vous attaquerez la seconde phase : la création du dictionnaire de mots.

Qu'est-ce que c'est, le « dictionnaire de mots » ?

C'est un fichier qui contiendra de nombreux mots pour votre jeu de Pendu. Il doit y avoir un mot par ligne. Exemple :

```
MAISON
BLEU
AVION
XYLOPHONE
ABEILLE
IMMEUBLE
GOURDIN
NEIGE
ZERO
```

À chaque nouvelle partie, votre programme devra ouvrir ce fichier et prendre un des mots au hasard dans la liste. Grâce à cette technique, vous aurez un fichier à part que vous pourrez éditer tant que vous voudrez pour ajouter des mots secrets possibles pour le Pendu.

Vous aurez remarqué que depuis le début je fais exprès d'écrire tous les mots du jeu en majuscules. En effet, dans le Pendu on ne fait pas la distinction entre les majuscules et les minuscules, le mieux est donc de se dire dès le début : « tous mes mots seront en majuscules ». À vous de prévenir le joueur, dans le mode d'emploi du jeu par exemple, qu'il est censé entrer des lettres majuscules et non des minuscules.
Par ailleurs, on fait exprès d'ignorer les accents pour simplifier le jeu (si on doit commencer à tester le é, le è, le ê, le ë… on n'a pas fini !). Vous devrez donc écrire vos mots dans le dictionnaire entièrement en majuscules et sans accents.

Le problème qui se posera rapidement à vous sera de savoir combien il y a de mots dans le dictionnaire. En effet, si vous voulez choisir un mot au hasard, il faudra tirer au sort un nombre entre 0 et X, et vous ne savez pas a priori combien de mots contient votre fichier.

Pour résoudre le problème, il y a deux solutions. Vous pouvez indiquer sur la première ligne du fichier le nombre de mots qu'il contient :

```
3
MAISON
BLEU
AVION
```

Cependant cette technique est ennuyeuse, car il faudra recompter manuellement le nombre de mots à chaque fois que vous en ajouterez un (ou ajouter 1 à ce nombre si vous êtes malins plutôt que de tout recompter, mais ça reste quand même une solution un peu bancale). Aussi je vous propose plutôt de compter automatiquement le nombre de mots en lisant une première fois le fichier avec votre programme. Pour savoir combien il y a de mots, c'est simple : vous comptez le nombre de **\n** (retours à la ligne) dans le fichier.

Une fois que vous aurez lu le fichier une première fois pour compter les\n, vous ferez un **rewind** pour revenir au début. Vous n'aurez alors plus qu'à tirer un nombre au sort parmi le nombre de mots que vous avez comptés, puis à vous rendre au mot que vous avez choisi et à le stocker dans une chaîne en mémoire.

Je vous laisse un peu réfléchir à tout cela. Ça va prendre plus ou moins de temps et c'est moins facile qu'il n'y paraît, mais en vous organisant correctement (et en créant suffisamment de fonctions), vous y arriverez.

Prend le temps de bien comprendre les sujets avant de les coder. Dans le monde professionnel en programmation, on passe 90% du temps de résolution du problème sur du papier à décrire le problème mathématiquement et ensuite les 10% sont utilisés pour le code. Encore une fois ne cherche pas de solution en ligne, tu en trouveras forcément mais tu n'auras pas avancé alors. Bon Courage 😉