TP n°2.5 – Introduction au C

1 Sans ordinateur

- **Q1.** On considère le code suivant, décrire l'état des variables (et des affichages) lors de l'exécution pour les valeurs $a = 3$ et $b = 4$.

```
void mystere (int a, int b){
  for(i=1;i<3;i+=1){
      b = b+i;
      for(int j=5; j<8; j+=1){
          a=a+i+j;
          printf("%d\n", a*b);
      }
  }
}
```

- **Q2.** On considère le code suivant, décrire les appels récursifs et leurs valeurs de retour lors de l'exécution pour les valeurs $n = 5$, $a = 2$ et $n = 3$, $a = 3$. Que fait cette fonction ?

```
int mystere (int n, int a){
  if(n==0){return 1;}
  else if(n%2==0){
      int b = mystere(n/2, a);
      return b*b;
  }
  else{int b = mystere(n/2, a);
       return b*b*a;}
}
```

2 Fibonacci

On rappelle que la suite de Fibonacci est définie ainsi :

$$F_0 = F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \forall n \geq 2$$

- **Q3.** Écrire une fonction qui calcule le n-ième terme de la suite de Fibonacci par récurrence.
- **Q4.** Écrire une fonction qui calcule le n-ième terme de la suite de Fibonacci en utilisant une boucle et le moins de variables possible.
- **Q5.** Tester pour de grands n (au-dessus de 50). Que se passe-t'il ? (Si vous ne savez pas, rappelez-vous que le type `int` est stocké sur 4 octets)

3 scanf

Il est possible de taper une valeur au clavier pour le programme en utilisant la commande **`scanf`**.

Pour cela : on crée une variable pour ranger ce qui est tapé au clavier (du type attendu), et on utilise **`scanf`**.

`scanf` fonctionne comme **`printf`**, avec une chaine de formattage, puis les variables où on range ce qui est tapé. On rajoute une esperluette devant le nom de la variable. Pour le moment, on ne fera cela que avec des int et des float.

Par exemple le code suivant permet de lire un entier x et un flottant y donnés au clavier. Recopiez dans `main`.

```
//étape 1
int x;
float y;

//étape 2
printf("Entrez un int et un float");
scanf("%d %f", &x, &y);
```

- **Q6.** Écrire un programme qui lit trois variables au clavier et affiche le maximum des trois
- **Q7.** Écrire un programme qui lit deux entiers a et b au clavier et demande à l'utilisateur d'écrire un entier `x` au clavier pour renvoyer :

- Si x=1 : savoir si $a+b$ est pair
- Si x=2 : savoir si $a*b$ est pair
- Si x=3 : connaitre le signe de $a+b$
- Si x=4 : connaitre le signe de $a*b$

- **Q8.** Écrire une fonction qui permet de jouer avec l'ordinateur au jeu de "plus chaud, plus froid"
  Pour faire de l'aléatoire, il faut d'abord écrire `#include <time.h>` en haut du fichier, puis `srand(time(NULL))` au début du `main`.
  L'ordinateur peut alors choisir une valeur aléatoire entre 0 et un certain n avec le code suivant :
  ```
  int r = rand() % n;
  ```
  Ensuite l'utilisateur propose des entiers au clavier et l'ordinateur répond si la valeur à deviner est plus grande ou plus petite, jusqu'à ce que l'utilisateur ait trouvé.

4 Sudoku

Pour les besoins de cette partie, on va prendre un tout petit peu d'avance sur le cours :

On peut créer une matrice de flottants de taille 10*34 avec l'instruction `float mat[10][34];`. Les valeurs qu'elle contient sont inconnues.

On peut accéder à la valeur de la case (i,j) d'une matrice avec `mat[i][j]`.

On peut modifier la valeur dans la case (i,j) de notre matrice avec `mat[i][j] = 3.0`.

- **Q9.** Créer la matrice $\begin{pmatrix} 3 & 5 & 6 \\ -10 & -7 & 3 \\ -7 & 3 & 78 \end{pmatrix}$ et la matrice de caractères $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$

Le sudoku est un jeu se jouant sur une grille $9*9$ séparée en 9 carrés de $3*3$ (on les appelera **sous-grilles**). Certaines cases sont déjà remplies. Le but est de remplir les autres avec des nombres entre 1 et 9 de sorte à ce que :

- Aucune colonne ne contienne deux fois la même valeur,
- Aucune ligne ne contienne deux fois la même valeur,
- Aucune sous-grille ne contient deux fois la même valeur

Notre but est d'écrire un algorithme qui trouvera la solution du sudoku, de la manière la plus naïve possible, c'est à dire en testant toutes les possibilités et en regardant ce qu'il se passe.

- **Q1.** Écrire une fonction `bool verifie_ligne(int mat[9][9], int ligne)` qui prend en entrée la matrice et un numéro de ligne et vérifie qu'aucune valeur n'apparait deux fois sur cette ligne.
- **Q2.** Faire la même chose avec une colonne.
- **Q3.** Faire la même chose avec les sous-grilles : on numérote les sous-grilles de gauche à droite de haut en bas et votre fonction doit fonctionner juste avec le numéro de la sous-grille (idéalement).
- **Q4.** Écrire une fonction `bool est_solution(int mat[9][9])` qui vérifie si une matrice représentant une grille de sudoku remplie vérifie toutes les règles du jeu.
- **Q5.** (Difficile) Écrire une fonction `void cherche_solution(int indices[20][3])` qui cherche une solution à une grille de sudoku dont 20 indices sont donnés.
  On prendra comme convention que pour chaque $i \in [|0, 19|]$, `indices[20][0]` est le numéro de la ligne de l'indice, `indices[20][1]` le numéro de la colonne de l'indice et `indices[20][2]` est la valeur de l'indice.
  Il vous faudra trouver un moyen d'énumérer toutes les grilles possibles.
  Attention : Vu le nombre de possibilités, testez cette fonction avec prudence. Vous pouvez à la limite tester votre énumération sur une matrice plus petite.

Si vous avez fini, allez faire le TP 2.75.