Devoir surveillé d'informatique

Consignes

- Les programmes demandés doivent être écrits en C et on suppose que les librairies standards usuelles (`<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`) sont déjà importées.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

❑ Exercice 1 : *exponentiation rapide*

On rappelle que pour $a \in \mathbb{R}, n \in \mathbb{N}^*$,

$$\begin{cases} a^n &= \left(a^{\frac{n}{2}}\right)^2, \text{ si } n \text{ est paire} \\ a^n &= \left(a^{\frac{n-1}{2}}\right)^2 \times a, \text{ sinon} \end{cases}$$

et d'autre part on convient que pour tout $a \in \mathbb{R}, a^0 = 1$.

1. Vérifier que pour calculer a^7, l'utilisation de ces relations ne demande que 4 multiplications, alors qu'il en faut 7 pour l'algorithme qui consiste à multiplier 1 par a à 7 reprises.
2. Ecrire une fonction récursive `exp_rapide` en C qui prend en argument un flottant `a` et un entier `n` et renvoie a^n en utilisant les relations de récurrence rappelées en début d'exercice.
3. **Bonus :** écrire cette fonction en OCaml.
4. Pour $n \in \mathbb{N}$, donner un ordre de grandeur (en le justifiant) du nombre de multiplication nécessaires pour calculer a^n à l'aide de la fonction `exp_rapide`.
5. Montrer que le nombre de multiplications nécessaire pour calculer a^{15} avec cet algorithme est 6.
6. Montrer (en donnant les étapes) qu'on peut calculer a^{15} en faisant seulement 5 multiplications. Que peut-on en conclure sur l'algorithme d'exponentiation rapide ?

❑ Exercice 2 : *palindrome*

Un **palindrome** est un mot qui se lit indifféremment de droite à gauche ou de gauche à droite, par exemple *kayak* est un palindrome, de même que *ressasser*. Par contre, *bébé* ou *eleve* ne sont pas des palindromes.

1. Ecrire, *sans utiliser* `strlen`, une fonction `longueur` qui prend en argument une chaine de caractère et renvoie sa longueur.
   🛟 On rappelle qu'une chaine de caractères est un tableau se terminant par le caractère spécial `'\0'`
2. Ecrire une fonction iterative `est_palindrome` qui prend en argument une chaine de caractères et renvoie `true` ou `false` suivant que cette chaine soit ou non un palindrome.
3. Ecrire une version récursive de cette fonction.
   🛟 On pourra éventuellement écrire une fonction auxiliaire `palindrome_rec` qui prend en argument une chaine de caractères `s` ainsi que deux entiers `debut` et `fin` et qui teste si la chaine de caractères démarrant à l'indice `debut` et se terminant à l'indice `fin` (inclus) est un palindrome.

❑ Exercice 3 : *convergence d'une suite*

On considère la suite :

$$\begin{cases} u_1 = 2 \\ u_2 = 3 \\ u_{n+2} = 15 - \dfrac{54}{u_{n+1}} + \dfrac{40}{u_n u_{n+1}} \end{cases}$$

1. Ecrire une fonction `calcule` qui prend en argument un entier `n`, ne renvoie rien et affiche dans le terminal les valeurs de u_k pour $k = 0, \ldots, n$ calculées à l'aide de la formule de récurrence ci-dessus. A titre d'exemple on donne ci dessous le résultat souhaité pour l'appel `calcule(10)` :

```
u0 = 2.000000
u1 = 3.000000
u2 = 3.666667
u3 = 3.909091
u4 = 3.976744
u5 = 3.994152
u6 = 3.998536
u7 = 3.999634
u8 = 3.999908
u9 = 3.999977
u10 = 3.999994
```

2. Ecrire un programme principal `main` qui prend en argument un entier sur la ligne de commande, et appelle la fonction `calcule` avec cet entier. A titre d'exemple, si votre programme est compilé sous le nom `suite.exe` alors `./suite.exe 10` doit produire l'affiche précédent.
   🛟 On rappelle que la fonction `atoi` permet de convertir une chaine de caractères en entier.
3. Montrer que le terme général de $(u_n)_{n\in\mathbb{N}}$ est $u_n = \dfrac{4^n+2}{4^{n-1}+2}$.
   🛟 Penser à faire un raisonnement par récurrence.
4. En déduire la limite de $(u_n)_{n\in\mathbb{N}}$.
5. On donne ci-dessous les 5 dernière lignes affichées par `./exercice1.exe 60`. Calculer la valeur exacte de u_{60} grâce à la formule établie à la question 3 et expliquer rapidement la différence avec la valeur calculée par le programme.

```
u56 = 9.999964
u57 = 9.999986
u58 = 9.999994
u59 = 9.999998
u60 = 9.999999
```

❑ **Exercice 4** : *Structures et pointeurs*

Dans un lycée les salles de cours sont nommées par une lettre suivie d'un numéro (par exemple `R4` ou `S6`). De plus ces salles ont une capacité maximale d'élèves donnée sous la forme d'un nombre entier et sont ou non équipées d'ordinateur.

1. Donner en C, la définition d'un type structuré `salle` contenant les champs suivants :
   — `batiment` de type `char`
   — `numero` de type `int`
   — `capacite` de type `int`
   — `ordinateur` de type `bool`

   Dans toute la suite de l'exercice on suppose ce type défini et nommé `salle`.
2. Ecrire en C, une fonction `cree_salle` qui prend en argument un caractère `b`, deux entiers `n` et `c` et un booléen `o` et renvoie une variable de type `salle` telle que `salle.batiment = b, salle.numero = n, salle.capacite = c` et `salle.ordinateur = o`.
3. Ecrire en C, une fonction `modifie_capacite` qui prend en argument une `salle s` et un entier `nc` qui affecte à cette salle la nouvelle capacité `nc`.
4. On suppose à présent qu'on dispose d'un fichier `salles.txt` contenant 100 lignes, sur chaque ligne on trouve le nom d'une salle suivie de sa capacité puis d'un `1` si la salle est équipée d'ordinateur et d'un `0` sinon. Par exemples voici les deux premières lignes du fichier `salles.txt` :

```
S6 30 0
R2 16 1
```

Ecrire une fonction `lire_salle` qui ne prend pas d'arguments, lit le fichier `salles.txt` et renvoie un tableau `salles` contenant les 100 salles du fichier. Par exemple le premier élément du tableau renvoyé `salles[0]` doit être tel que `salles[0].batiment = 'S', salles[0].numero = 6, salles[0].capacite = 30` et `salles[0].ordinateur = false`.

5. Ecrire une fonction `capacite_batiment` qui prend en argument un tableau de `salles` (ainsi que sa taille) et un caractère `bat` et renvoie la capacité totale des salles dont le batiment est `bat`.

❑ **Exercice 5** : *Implémentation des entiers par représentation binaire*

On rappelle qu'en C, le type `uint64_t` (disponible dans `stdint.h` qu'on suppose déjà importée dans la suite de l'exercice) représente des entiers *non signés* sur 64 bits. D'autre part on rappelle que le spécificateur de format permettant d'afficher un entier de type `uint64_t` est `%lu`.

1. A propos du format `uint64_t`.
   a) Donner l'intervalle d'entiers représentable avec ce format.
   b) En compilant puis en exécutant le programme suivant sur un ordinateur :

```c
#include <stdio.h>
#include <stdint.h>

int main()
{
    uint64_t a=0;
    a = a - 1;
    printf("a= %lu\n",a);
}
```

   on a obtenu l'affichage suivant dans le terminal : `a= 18446744073709551615`. Expliquer cet affichage, s'agit-il d'un comportement indéfini ?
   c) Convertir $\overline{11101011}^2$ en base 10.
   d) Convertir $\overline{307}^{10}$ en base 2.
2. Représentation des ensembles.
   On utilise à présent les entiers au format `uint64_t` afin de représenter des ensembles. A chaque entier écrit en base 2 on associe l'ensemble dont les éléments sont les positions des bits égaux à 1. Par exemple :
   — L'entier $\overline{11001}^2 (=\overline{25}^{10})$ a des bits égaux à 1 aux positions 0,3 et 4 et donc représente l'ensemble $\{0, 3, 4\}$.
   — L'entier $\overline{10000000}^2 (=\overline{128}^{10})$ a un seul bit égal à 1 en position 7 et donc représente l'ensemble $\{7\}$.
   — L'ensemble $\{1, 5\}$ est représenté par l'entier ayant des bits égaux à 1 en position 1 et 5, c'est à dire $\overline{100010}^2 = \overline{34}^{10}$.
   a) Quels sont les ensembles représentables avec ce codage avec des entiers au format `uint64_t` ?
   b) Donner l'écriture en base 10 de l'entier représentant l'ensemble $\{2, 7\}$
   c) Quel est l'ensemble codé par l'entier $\overline{76}^{10}$?
   d) Donner la caractérisation des ensembles représentés par une puissance exacte de 2.
   e) Ecrire une fonction `appartient` qui prend en argument un entier `s` codant un ensemble et un entier `e` et renvoie `true` si `e` appartient à l'ensemble codé par `s` et `false sinon`. Par exemple puisque l'ensemble $\{1, 5\}$ est codé par 34, `appartient(34,1)` doit renvoyer `true` tandis que `appartient(34,2)` doit renvoyer `false`.
   f) Ecrire une fonction `encode` en C, qui prend en argument un tableau d'entiers (et sa taille) et renvoie l'entier qui représente l'ensemble dont les éléments sont ceux du tableau. On supposera que les éléments du tableau sont distincts et tous inférieurs à 63. Par exemple, si `tab` est un tableau de taille 2 tel que `tab[0]=1` et `tab[1]=5` alors, `encode(tab,2)` doit renvoyer $\overline{100010}^2$ c'est à dire $\overline{34}^{10}$.
   g) Ecrire une fonction `decode` en C, qui prend en argument un entier au format `uint64_t` et renvoie l'ensemble qu'il représente sous la forme d'un tableau. Par exemple `decode(34)` doit renvoyer un tableau `tab` tel que `tab[0]=1` et `tab[1]=5`.