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 : *chercher les erreurs*

1. Les fonctions ci-dessous contiennent une ou plusieurs erreurs et/ou ne respectent pas leurs spécifications (données en commentaire). Dans chaque cas, expliquer les erreurs commises et les corriger.

a) Fonction `harmonique`

```c
// Calcule la somme des inverses des entiers jusqu'à n
// Par exemple harmonique(3) renvoie 1 + 1/2 + 1/3
double harmonique(int n){
    double sh = 0;
    for (int i = 0; i < n; i = i + 1)
    {
        sh = sh + 1 / i;
    }
    return sh;}
```

b) Fonction `tous_egaux`

```c
// Renvoie true si tous les éléments de tab sont identiques
// Par exemple si test={2, 2, 2, 2} alors tab(test,4) renvoie true
bool tous_egaux(int tab[], int size){
    for (int i = 0; i < size; i = i + 1)
    {
        if (tab[i] != tab[i + 1])
        {
            return false;
        }
        else
        {
            return true;
        }
    }
}
```

c) Fonction `cree_tab_entiers`

```c
int *cree_tab_entiers(int n){
    int tab_entiers[n];
    for (int i = 0; i < n; i++)
    {
        tab_entiers[i] = i;
    }
    return &tab_entiers;}
```

d) Fonction `syracuse`

```c
void syracuse(int n){
    if (n % 2 = 0)
    {
        n = n / 2;
    }
    else
    {
        n = 3 * n + 1;
    }}
```

2. Les programmes suivants, produisent une erreur à l'exécution. Expliquer l'origine du problème et apporter les corrections nécessaires

a) Affichage d'une adresse

```c
// Programme qui affiche l'adresse de la variable p
int main(){
    int *p;
    *p = 42;
    printf("%p", p);}
```

b) Calcul de la somme de deux entiers

```c
// Programme qui demande deux entiers puis affiche leur somme
int main(){
    int a, b;
    printf("a=");
    scanf("%d", a);
    printf("b=");
    scanf("%d", b);
    printf("Somme = %d", a + b);}
```

❑ Exercice 2 : *Pointeurs*

1. Compléter le tableau suivant, qui donne l'état des variables au fur et à mesure des instructions données dans la première colonne (on a indiqué par ✖ une variable non encore déclaré.)

instructions	a	b	p	q
`int a = 14;`	14	✖	✖	✖
`int b = 42;`	14	...	...	...
`int *p = &a;`	14	...	&a	...
`int *q = &b;`	14	...	&a	...
`*p = *p + *q ;`	...	...	...	...
`*q = *p - *q ;`	...	...	...	...
`*p = *p - *q ;`	...	...	...	...

2. Ecrire une fonction en C qui prend en argument deux pointeurs vers des entiers, ne renvoie rien et échange les valeurs de ces deux entiers *sans utiliser de variable temporaire*.
3. Compléter le programme suivant en écrivant l'appel à la fonction `echange` afin d'échanger les valeurs des entiers `n` et `m`

```
    int n = 55, m = 12;
    ..........
```

❑ Exercice 3 : *puissance*

1. Ecrire une fonction `valeur_absolue` qui prend en argument un entier n et renvoie sa valeur absolue $|n|$.
   On rappelle que : $|n| = \begin{cases} -n & \text{si } n < 0 \\ n & \text{sinon} \end{cases}$
2. Ecrire une fonction `puissance` qui prend en argument un flottant (type `double`) a et un entier n et renvoie a^n. On rappelle que pour $a \in \mathbb{R}^*$, $n \in \mathbb{Z}$:
   $$\begin{cases} a^n = \underbrace{a \times \cdots \times a}_{n \text{ facteurs}} & \text{si } n > 0, \\ a^0 = 1, & \\ a^n = \dfrac{1}{a^{-n}} & \text{si } n < 0. \end{cases}$$
   D'autre part $0^0 = 1$, $0^n = 0$ si $n > 0$ et les puissances négatives de zéro ne sont pas définies. On vérifiera la précondition $n > 0$ lorsque $a = 0$ à l'aide d'une instruction `assert`.
3. Tracer le graphe de flot de contrôle de cette fonction.
4. Proposer un jeu de test permettant de couvrir tous les arcs.

❑ Exercice 4 : *Annagrammes*

Deux mots *de même longueur* sont anagrammes l'un de l'autre lorsque l'un est formé en réarrangeant les lettres de l'autre. Par exemple :

— *niche* et *chien* sont des anagrammes.

— *epele* et *pelle*, ne sont pas des anagrammes, en effet bien qu'ils soient formés avec les mêmes lettres, la lettre *l* ne figure qu'à un seul exemplaire dans *epele* et il en faut deux pour écrire *pelle*.

Le but de l'exercice est d'écrire une fonction en C qui renvoie `true` si les deux chaines données en argument sont des anagrammes et `false` sinon. On suppose que les chaines sont constitués uniquement de lettres majuscules.

1. Ecrire une fonction `tableau_egaux` qui prend en argument deux tableaux ainsi que leur taille et renvoie `true` lorsque ces deux tableaux ont un contenu et une longueur identique et `false` sinon.
2. Ecrire une fonction `nb_lettres` qui prend en argument une chaine de caractères `chaine` et renvoie un tableau d'entiers `tab` de longueur 26 de sorte que `tab[i-1]` contienne le nombre de fois où la ième lettre de l'alphabet apparait dans `chaine`. Par exemple tab[0], doit contenir le nombre de fois où la lettre *a* apparait dans le mot.
3. En supposant les deux fonctions précédentes correctement écrites, on propose le code suivant pour la fonction `anagrammes` qui teste si les deux chaines données en argument sont des anagrammes :

```
bool anagrammes(char * chaine1, char *chaine2)
{
   return tab_egaux(nb_lettres(chaine1),26,nb_lettres(chaine2),26);
}
```

   Cette fonction renvoie le résultat attendu mais pose un problème, lequel ? Expliquer et proposer une correction.

❑ Exercice 5 : *tri à bulles*

Le tri à bulles est un algorithme de tri qui parcourt le tableau à l'aide d'un indice i de la fin vers le début. Pour chacun de ces indices i, on parcourt la partie du tableau allant de l'indice 0 à l'indice $i-1$ et si les deux éléments consécutifs situés aux indices j et $j+1$ ne sont pas dans l'ordre croissant, on les échange. Par exemple sur le tableau `{7, 2, 9, 5, 3}`

- $i = 4$ (on rappelle que i parcourt de la fin vers le début)
  — $j = 0$, donc on échange `tab[0]` et `tab[1]` car ils ne sont pas dans l'ordre croisant : `{2, 7, 9, 5, 3}`

— $j = 1$, pas d'échange (7 et 9 sont en ordre croissant)

— $j = 2$, échange car 9 et 5 ne sont pas dans l'ordre croissant : `{2, 7, 5, 9, 3}`

— $j = 3$, échange et on obtient `{2, 7, 5, 3, 9}`

- $i = 3$, cette fois on parcourt avec $j = 0$ jusqu'à 2, en effet à l'étape précédente le plus grand élément du tableau se retrouve forcement en dernière position. On obtient en fin de parcours : `{2, 5, 3, 7, 9}`
- $i = 2$, à la fin de cette itération on obtient `{2, 3, 5, 7, 9}`

1. Faire fonctionner cet algorithme à la main sur le tableau `{11, 2, 5, 13, 8, 4}` et donner l'état du tableau à la fin de chaque itération de l'indice j pour j variant de 0 à i en recopiant et complétant le tableau suivant :

i	valeurs contenu dans le tableau à la fin de l'itération d'indice i.
5	.....
4	.....
3	.....
2	.....
1	.....

2. Donner un exemple de tableau de longueur 5, *non trié initialement* qui sera entièrement trié après le premier tour de boucle de l'indice j (c'est à dire pour $i = 4$).
3. Donner un exemple de tableau qui ne sera trié qu'à la fin de toutes les itérations de l'indice i.
4. On veut maintenant écrire cet algorithme en C, en effectuant le tri dans une copie du tableau. On suppose déjà écrite la fonction `echange` qui prend en argument un tableau et deux indices ne renvoie rien et échange les éléments situés à ces deux indices.
   a) Ecrire un fonction `copie_tab` qui prend en argument un tableau ainsi que sa taille et renvoie un pointeur vers une copie de ce tableau
   b) Ecrire une fonction `tri_bulles` qui prend en argument un tableau `tab` ainsi que sa taille, ne modifie pas ce tableau et renvoie un pointeur vers un tableau contenant les éléments du tableau `tab` triés dans l'ordre croissant.
   c) Afin de tester cette fonction on a écrit le programme principal suivant :

```c
int main()
{
    int tab[6] = {5, 11, 8, 9, 4, 6};
    int *tab_trie = trie_bulles(tab,6);
    for (int i=0; i<6; i++)
    {printf("%d ", tab_trie[i]);}
    printf("\n");
}
```

   Quelle instruction est manquante dans ce programme ? Quelle option de compilation signalerait le problème lors de l'exécution ?