Informatique *TD*
Programmation impérative avec C - Pointeurs, tableaux, structures Ipesup

1 Soupe à la tomate

C'est la saison de la soupe à la tomate ! En fin gourmet que vous êtes, vous ajoutez des pâtes en forme de lettres à votre soupe. Ces lettres, à leur tour, forment des phrases.

Vous remarquez que vous pouvez écrire des phrases rigolotes en mangeant une lettre de temps en temps.

On vous demande d'écrire deux fonctions : `manger` et `soupe`.

La fonction `manger` prend deux arguments :

— une chaîne de caractères `texte`, représentée par un pointeur de type `char *` ;
— un entier `n`.

Elle doit renvoyer une nouvelle chaîne de caractères correspondant à `texte` après avoir supprimé une lettre toutes les `n` positions. La première lettre de `texte` est toujours mangée (conservée).

La fonction `soupe` ne prend aucun argument. Elle doit :

1. demander à l'utilisateur d'entrer un texte (une phrase sans espace, ou utiliser `fgets`) ;
2. demander à l'utilisateur un nombre entier `n` ;
3. appeler la fonction `manger` sur le texte et le nombre saisis ;
4. afficher le résultat.

Exemple d'exécution attendue :

```
Entrez un texte : BONJOUR
Entrez un nombre : 3
Résultat : BOOUR
```

Indications :

— Utilisez les pointeurs pour parcourir la chaîne (`*(texte + i)` ou `texte[i]`).
— Vous pouvez allouer une nouvelle chaîne pour le résultat avec `malloc`.
— N'oubliez pas de terminer la chaîne par le caractère nul `'\0'`.

2 Mon dragon préféré

On souhaite écrire une fonction `mon_dragon_prefere` qui prend en argument les caractéristiques d'un Dragon : la couleur de ses écailles, la couleur de ses yeux, sa taille (en mètres) et le nom de son souffle.

La fonction aura donc quatre arguments, tous de type `char *` :

- `couleur_ecaille` par exemple : `"Violet"`, `"Bleu"`, `"Blanc"`, etc.
- `couleur_yeux` par exemple : `"Rouge"`, `"Orange"`, `"Marron"`, etc.
- `taille` par exemple : `"37"`, `"56"`, `"58"`, etc.
- `nom_souffle` par exemple : `"Vent du Désert"`, `"Flamme de Magma"`, etc.

Les deux derniers arguments (`taille` et `nom_souffle`) sont optionnels : par défaut, ils valent `"-"` et ne doivent être affichés que si leur valeur est différente de `"-"`.

La fonction doit afficher à l'écran toutes les caractéristiques du dragon sous la forme suivante :

```
Fuyez, voici le Dragon <couleur_ecaille> aux Yeux <couleur_yeux> !
```

et, si la taille est précisée :

```
Quelle taille ! Il mesure <taille> m !
```

et, si le souffle est précisé :

```
Son souffle est appelé : <nom_souffle> !
```

Exemples d'exécution :

```
mon_dragon_prefere("Vert", "Violets", "-", "-");
→ Fuyez, voici le Dragon Vert aux Yeux Violets !
```

```
mon_dragon_prefere("Orange", "Argentés", "37", "-");
→ Fuyez, voici le Dragon Orange aux Yeux Argentés !
  Quelle taille ! Il mesure 37 m !
```

```
mon_dragon_prefere("Bleu", "Rouges", "42", "Flamme de Magma");
→ Fuyez, voici le Dragon Bleu aux Yeux Rouges !
  Quelle taille ! Il mesure 42 m !
  Son souffle est appelé : Flamme de Magma !
```

Indications :

- Inclure la bibliothèque `<string.h>` pour utiliser `strcmp`.
- Tous les arguments sont des chaînes de caractères (même la taille).
- Vous devez utiliser des pointeurs dans la définition de la fonction :
  `void mon_dragon_prefere(char *ecaille, char *yeux, char *taille, char *souffle).`

3 Opérations sur tableaux (C, pointeurs)

Dans toutes les fonctions ci-dessous, le tableau d'entrée `l` **ne doit pas être modifié**.
On manipulera des pointeurs et on allouera les nouveaux tableaux avec `malloc`.
On utilisera le couple `(int *tab, size_t n)` pour représenter un tableau d'entiers et sa taille.

Rappel C. Une fonction qui construit un nouveau tableau renverra `int *` et écrira la taille de sortie dans un paramètre `size_t *out_n`. Le code appelant est responsable de `free`.

1. **`decale`**.
   Écrire `int *decale(const int *l, size_t n, int d, size_t *out_n);`
   qui renvoie un nouveau tableau obtenu en ajoutant un décalage `d` à chaque élément de `l`.
   Ex. `decale([4,17,12],3,3)` → `[7,20,15]`.
2. **`intercale_zeros`**.
   Écrire `int *intercale_zeros(const int *l, size_t n, size_t *out_n);`
   qui renvoie le tableau où l'on intercale un `0` après chaque élément. La taille de sortie vaut `2*n`.
   Ex. `intercale_zeros([4,17,12])` → `[4,0,17,0,12,0]`.
3. **`supprime`**.
   Écrire `int *supprime(const int *l, size_t n, int elem, size_t *out_n);`
   qui renvoie le tableau obtenu en supprimant *toutes* les occurrences de `elem`.
   Ex. `supprime([4,7,12,4,4,0,4,5],8,4)` → `[7,12,0,5]`.
4. **`insere_milieu`**.
   Écrire `int *insere_milieu(const int *l, size_t n, int elem, size_t *out_n);`
   qui insère `elem` "au milieu" de `l` :
   — si `n` est pair, on insère `elem` une fois à l'indice `n/2` ;
   — si `n` est impair, on insère `elem` *de part et d'autre* de l'élément central (deux insertions).
   La taille de sortie vaut `n+1` ou `n+2`.
   Ex. `insere_milieu([4,7,12,3],4,0)` → `[4,7,0,12,3]`
   `insere_milieu([9,3,5,6,2],5,1)` → `[9,3,1,5,1,6,2]`.
5. **`transpose`**.
   Écrire une fonction qui transpose une matrice d'entiers de taille `n × m`. La matrice d'entrée est représentée par un tableau de pointeurs :

   ```
   int **transpose(const int **mat, size_t n, size_t m);
   ```

   La fonction renvoie une nouvelle matrice de taille `m × n`, allouée avec `malloc`, telle que `res[j][i] = mat[i][j]` pour tout `i, j`.
   Exemple :

   `mat = [[1,2,3],[4,5,6]]` → `res = [[1,4],[2,5],[3,6]]`.

6. **`diagonale_principale`**.
   Écrire une fonction qui extrait la diagonale principale d'une matrice carrée `n × n`.

   ```
   int *diagonale_principale(const int **mat, size_t n);
   ```

   La fonction renvoie un nouveau tableau de taille `n`, contenant les éléments `mat[i][i]`.
   Exemple :

   `mat = [[2,4,7],[0,5,8],[1,9,3]]` ⇒ `[2,5,3]`.