Exercices: Série 5 - Chaînes de caractères - ICC-C 2025-2026

1. Échauffement : longueur d'une chaîne de caractères

Rappelez-vous qu'une chaîne de caractères (ou *string*) est un tableau de `char`, qui est terminée par le caractère spécial `\0`. Ce caractère de fin n'est pas considéré comme faisant partie de la chaîne. La chaîne stockée sous la forme `{ 'f', 'o', 'o', '\0' }` contient donc 3 caractères : `f`, `o` et `o`. Sa longueur est 3.

Écrire votre propre fonction `my_strlen`, qui calcule la longueur d'une chaîne de caractère. Elle doit prendre un tableau de caractères en entrée, et renvoyer la longueur. Tester votre fonction avec quelques chaînes.

Réflexion :

- Le paramètre devrait-il avoir le mot-clef `const` ?
- Quel est le type de retour de votre fonction ?
- Que se passe-t-il si on a oublié le caractère de fin `\0` ?

Découverte :

- Quelle longueur est calculée pour la chaîne `"Sébastien"` ? Le résultat vous paraît-il correct ?
- Qu'en est-il de la chaîne `"👫"` ?

2. Occurrences

On cherche à savoir combien de fois une sous-chaîne (*substr*) apparaît à l'intérieur d'une autre chaîne (*str*). Par exemple, la sous-chaîne `"est"` apparaît une seule fois dans la chaîne `"estuaire"`.

Les occurrences peuvent se chevaucher. `"aba"` apparaît 3 fois dans `"abababa"`.

Écrire une fonction `a_sous_chaine` qui prend 3 paramètres : des chaînes de caractères `substr` et `str`, ainsi qu'un indice `p`. Elle doit renvoyer `true` si on trouve `substr` dans `str` à partir de la position `p`. Par exemple :

- `a_sous_chaine("foo", "foo babar", 4) == false`
- `a_sous_chaine("baba", "foo babar", 4) == true`
- `a_sous_chaine("baba", "foo babar", 1) == false`
- `a_sous_chaine("baba", "foo babar", 7) == false`

Puis, écrire une fonction `occurrences` qui prend deux chaînes de caractères en paramètres, `substr` et `str`. Elle renvoie le nombre d'occurrences de `substr` dans `str`.

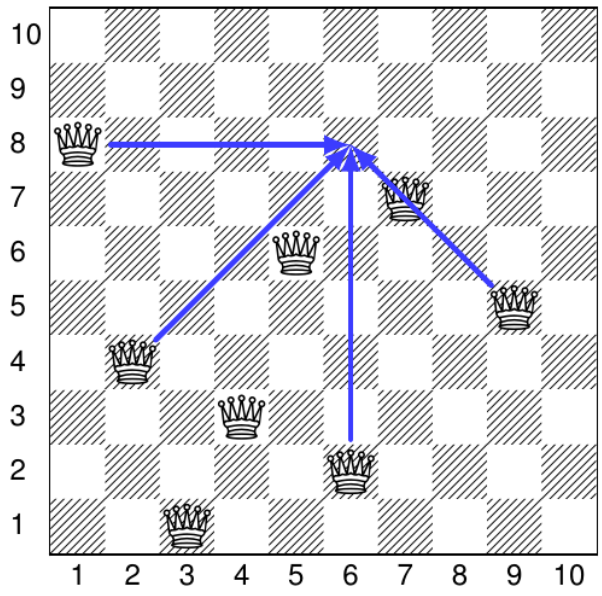
Testez d'abord vos fonction avec quelques exemples (comme ceux donnés plus haut).

Puis, depuis `main()`, lisez ces deux chaînes sur l'entrée standard. D'abord `substr` sur une ligne, puis `str` sur une deuxième ligne. Utilisez `fgets(dest, taille_max, stdin)`. On supposera que les deux chaînes ne font pas plus de 256 caractères.

> Bon à savoir : vous pouvez utiliser l'entrée standard pour lire un fichier plutôt que lire le clavier. Pour ce faire, lancez votre programme avec `./monprog < fichier.txt`. Cela peut vous aider à tester plus vite votre programme, car vous pouvez préparer toutes les entrées dans un fichier.

3. La menace des reines

Sur un échiquier de taille 10×10, on place k reines. Pour une position donnée, il faut indiquer combien de reines attaquent cette position. Si deux reines sont alignées avec la position, on compte les deux. (Une reine attaque toutes les cases qui sont sur sa ligne, sur sa colonne, et sur les deux diagonales qui se croisent à sa position.)



Dans cet exemple, on a placé 8 reines. Selon les règles, la position au croisement de la ligne 8 et de la colonne 6 est attaquée par 5 reines : les reines aux positions $(8,1)$, $(4,2)$, $(2,6)$, $(7,7)$, $(5,9)$.

L'entrée contient d'abord deux entiers r et c (entre 1 et 10 inclus) représentant la ligne et la colonne de la position qui nous intéresse. Ensuite vient l'entier k (le nombre de reines) suivi par k paires d'entiers indiquant les positions des k reines.

Pour l'exemple ci-dessus, l'entrée contiendrait (à mettre dans un fichier et à lire avec `<`) :

```
8 6
8
8 1
4 2
1 3
3 4
6 5
2 6
7 7
5 9
```

Écrire un programme qui lit cette entrée et affiche le nombre de reine qui attaquent la position donnée. Dans l'exemple, le programme doit afficher `5`.

Conseil : Vous voudrez sans doute lire les lignes et colonnes des reines dans deux tableaux séparés, chacun de taille k.

Indice : Avez-vous besoin de stocker tout l'échiquier ?

Pouvez-vous généraliser votre programme pour qu'il fonctionne avec un échiquier de taille $N \times N$? Il faudra peut-être ajouter la valeur de N dans l'entrée du programme.