IN 101 - TD 01

Énoncé

Dans les exercices, lorsqu'il vous est demandé d'écrire une fonction puis de la tester, implicitement vous devrez écrire un `main` qui permet d'appeler et tester cette fonction. Vous pourrez récupérer les entrées nécessaires pour tester la fonction en utilisant `scanf` (vous n'avez pas encore vu comment interagir avec les arguments de ligne de commande).

1 Caractère d'une chaîne

Q 1.1. Écrivez une fonction `nth` prenant en argument une chaîne de caractères et une position et **affichant** le caractère se trouvant à cette position dans la chaîne.

Rajoutez ensuite un `main` qui demande à l'utilisateur d'entrer au clavier une chaîne et une position (en utilisant `scanf`).

Q 1.2. Testez votre programme avec les entrées `"foo"`, 1. Cette chaîne permet-elle de faire un test intéressant ?

Q 1.3. Testez votre programme avec `"aloha"`, -10. Que révèle ce test ? Modifiez votre code si nécessaire.

Q 1.4. Testez votre programme avec `"aloha"`, 100000. Que révèle ce test ? Modifiez votre code si nécessaire.

Q 1.5. Testez à nouveau votre programme avec `"aloha"`, -10 et interprétez le résultat.

Q 1.6. On souhaite maintenant que la fonction **retourne** le caractère plutôt qu'elle ne l'affiche. Quel problème se pose à vous ? Proposer des solutions.

2 Polynôme

Q 2.1. Écrivez une fonction `poly` prenant en argument un nombre x et retournant la valeur du polynôme $5x^3 + 2x - 3x^4 + 5$ évalué en ce nombre.

Attention : n'utilisez pas la fonction `pow` de la bibliothèque standard qui masque la réalité abordée dans cet exercice.

Rajoutez ensuite un `main` qui demande à l'utilisateur d'entrer au clavier un réel (flottant) (en utilisant `scanf`).

Q 2.2. Combien d'opérations votre programme effectue-t-il ?

Q 2.3. Comment pouvez-vous réécrire votre algorithme pour diminuer ce nombre ?

3 Colinéarité

On souhaite écrire une fonction prenant les coordonnées de 3 points **dans le plan** et **retournant** une valeur disant si ces 3 points sont alignés.

Q 3.1. Quels sont les domaines des entrées et des sorties de cette fonction ?

Q 3.2. Sachant que si les points A, B et C sont alignés alors $\overrightarrow{AB}$ et $\overrightarrow{BC}$ sont colinéaires, donc $\overrightarrow{AB} = k\,\overrightarrow{BC}$, imaginez un algorithme et écrivez la fonction `col` qui réalise ce calcul.

Q 3.3. Testez votre programme, regardez par exemple pour `col (3, 3, 3, 3, 3, 3)`.

Q 3.4. Si l'on a rencontré le problème illustré en Q3.3, réécrire le programme pour l'éviter.

Q 3.5. Une fonction n'est pas toujours appelée avec des constantes. Elle peut faire partie de calculs plus complexes. Dans votre `main`, déclarez une variable `x` valant 13.1 et une variable `y` valant $\sqrt{\texttt{x}}$. Testez votre fonction, en l'appelant avec `(1, 1, y * y, x, 5, 5)`.

Remarque : la fonction $\surd$ en `C` s'appelle `sqrt` et vous devez importer les fonctions mathématiques par : **#include** <math.h>. Sur la ligne de compilation, rajoutez l'option `-lm` en fin de ligne de commande :

```
gcc -Wall col.c -lm
```

Q 3.6. Reprenez votre fonction pour corriger le problème identifié (si vous avez eu le problème) et testez son fonctionnement.

4 Somme à partir de mêmes chiffres

Q 4.1. Écrivez une fonction `same_sum` prenant en argument un entier n et un chiffre c entre 0 et 9 inclus et vérifiant si n est égal à une somme finie de c.

Q 4.2. Testez votre programme, par exemple avec same_sum (80, 40). Que doit-il répondre ?

Q 4.3. Proposez et implémentez une solution.

5 Chaîne d'un nombre + 1

Q 5.1. Écrivez une fonction `changed_digit` prenant en argument 1 chaîne de caractères représentant un nombre écrit en base 10, et dit si le chiffre de poids le plus fort (donc le plus à gauche dans l'écriture) changera si l'on ajoute 1 à ce nombre.

Par exemple, `changed_digit ("123")` renverra ≪faux≫ alors que `changed_digit ("19")` renverra ≪vrai≫.

Attention : on considère par hypothèse que la chaîne représente bien un entier (ne contient pas de caractères illégaux) et n'est donc forcément pas vide. Pas la peine de faire ces vérifications dans votre algorithme.

Attention : on considère qu'il n'y a pas de 0 non significatifs en tête.

Q 5.2. Testez votre programme avec `changed_digit ("-10")`. Testez-le avec `changed_digit ("+19")`. Qu'en déduisez-vous ?

Q 5.3. Si besoin, corrigez votre algorithme.

6 Somme de multiples

Q 6.1. Écrivez une fonction `sum` prenant en argument 2 bornes entières `x` et `y` et retournant la somme des nombres **multiples de 5** qui ne sont **pas pairs** entre ces bornes **incluses**.

S'il vous reste du temps ou pour continuer après la séance.

7 Différences de paires

Q 7.1. Écrivez une fonction `count` prenant en argument un tableau `t` contenant des entiers, la taille de ce tableau et une valeur entière `n`. Cette fonction doit **afficher** le **nombre** de couples formés d'éléments de `t` dont la différence des composantes vaut `n`.

Le tableau sera directement défini statiquement (≪en dur≫) dans votre `main` (donc vous connaîtrez sa longueur).

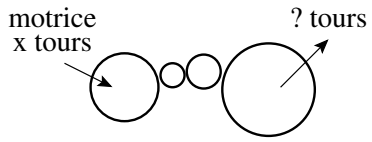
Un squelette de programme (`diffcouples_skel.c`) vous est donné avec un `main` contenant les tests proposés dans la question suivante.

Q 7.2. Testez votre programme avec au moins les cas suivants et vérifiez si vous obtenez les résultats attendus.

```
count ([1, 2, 3, 4], 1)       → 3
count ([1, 2, 3, 4], 0)       → 4
count ([1, 2, 3, 4], -1)      → 3
count ([0, 0, 0, 0], 0)       → 16
count ([-1, -1, -1, -1], -1)  → 0
```

8 Train d'engrenages

On considère un train d'engrenages (ou roues idéalement collantes) où chaque engrenage est en contact avec un seul suivant. Un tel train est représenté par un tableau contenant le périmètre de chaque engrenage. La première case du tableau contient le périmètre de l'engrenage moteur (que l'on va faire tourner de x tours).



[p1, p2, p3, p4]

Q 8.1. Écrivez une fonction `turns` prenant en argument un nombre de tours à appliquer au premier engrenage de la chaîne et retournant le nombre de tours effectués par le dernier engrenage de la chaîne.

9 Une seule fonction...

Q 9.1. Écrivez un programme qui affiche les arguments de ligne de commande qu'il a reçus.

Attention : pour faire cet exercice, il est indispensable de savoir comment l'on récupère les arguments de la ligne de commande. Si ce n'est pas votre cas, demandez à votre enseignant ou prenez un peu d'avance sur IN102 en lisant le paragraphe du polycopié qui explique ce point.

Contrainte : votre programme **ne devra pas** utiliser de boucles et ne devra comporter que **1 seule** fonction (qui elle, a le droit d'appeler ce qu'elle veut). De même, vous n'avez évidemment pas le droit à l'instruction diabolique `goto` (que nous n'avons pas vue et que nous ne verrons pas car c'est la porte d'entrée à des programmes incompréhensibles et impossibles à maintenir).

Si cela peut vous simplifier, vous pouvez afficher les arguments en ordre inverse de celui sur la ligne de commande.