I2010 : langage C (TP6)

Écriture de fonctions

Rappel : *En langage C, le passage des paramètres à une fonction se fait toujours PAR VALEUR !*

Testez systématiquement vos solutions avec valgrind.

1. *Implémentation de fonctions*

1.1. *Fonction booléenne*

- Concevez une fonction qui indique si un nombre entier est premier[1]
  - PRE : n : nombre entier positif
  - RES : vrai si n est premier ; faux sinon
  - `bool isPrime (int n)`

1.2. *Fonction renvoyant un tableau*

- Concevez une fonction
  - PRE : n >= 2
  - POST : sz est égal à la taille (physique et logique) du tableau renvoyé
  - RES : un tableau trié contenant tous les nombres premiers plus petits que n si aucune erreur n'est survenue ; NULL si une erreur est survenue (sz est mis à 0)
  - `int* prime_numbers (int n, int* sz)`
- Concevez un programme qui teste votre fonction en affichant les 25 nombres premiers inférieurs à 100.

1.3. *Fonction de manipulation de tableau*

- Concevez une fonction
  - PRE : t : tableau de longueur sz
  - POST : t représente un tableau trié contenant les sz premiers nombres premiers.
  - RES : vrai en cas de succès ; faux sinon
  - `bool first_prime_numbers (int* t, int sz)`

Pour concevoir cette fonction, le plus simple est de réutiliser la fonction `prime_numbers()` de la manière suivante (attention le code n'est pas complet) :

[1] https://fr.wikipedia.org/wiki/Nombre_premier

```
int n = 2;
int szr;
int* pn = prime_numbers(n, &szr);
while (szr < sz) {
  n = n * 2;
  pn = prime_numbers(n, &szr);
}
// copier les sz premiers entiers de pn dans t
return true;
```

- Complétez le programme précédant pour qu'il teste votre fonction en affichant les 100 premiers nombres premiers.

2. *Code refactoring pour améliorer la lisibilité et la robustesse d'un programme*

Reprenez le 2ième programme du TP 5 : « Chaîne de caractères et fonction "fgets" ».

2.1. *Fonction qui reçoit des paramètres et ne renvoie aucune valeur*

- Ajoutez une fonction
  - PRE : `t` : tableau de chaînes de caractères de taille `sz` ne contenant pas de chaînes nulles.
  - POST : Les chaînes de caractères de `t` sont affichées sur stdout dans l'ordre des indices à raison d'une chaine par ligne.
  - `void impTable (char** t, int sz)`
- Modifiez le programme de telle manière qu'il fonctionne comme suit :
  - il récupère les arguments sur la ligne de commande,
  - affiche la table des chaînes de caractères grâce à la fonction `impTable`
  - tant que la fin de fichier n'est pas rencontrée
    - lit un mot d'au plus 256 caractères
    - regarde si le mot est contenu dans la table
    - affiche le mot suivi de « présent » ou « absent » suivant le cas
  - affiche le nombre de fois qu'un mot lu n'a pas été trouvé dans la table.

2.2. *Fonction qui reçoit des paramètres et renvoie une valeur*

Ecrivez une fonction

- PRE : `inv` : chaîne de caractères
  `s` : tableau de char de taille `sz`
- POST : La fonction a affiché le message d'invitation `inv` sur stdout, ensuite elle a lu sur stdin une chaîne de caractère d'au plus sz – 1 caractères, a vérifié qu'elle n'est ni vide, ni trop longue et répété l'opération tant qu'une chaîne valide n'est pas introduite, puis elle a remplacé le '\n' par '\0'. La chaîne valide a été placée dans s.

- RES : renvoie le nombre de caractères de la dernière chaîne lue ; -1 si fin de fichier [Ctrl-D] atteinte
- `int litEtValideChaine (char* inv, char* s, int sz)`

En résumé, la fonction identifie et traite successivement 3 cas de chaînes invalides :

1) fin de fichier ➔ renvoie -1
2) chaîne vide ➔ lit une nouvelle chaîne sur *stdin*
3) chaîne trop longue ➔ vide le buffer et lit une nouvelle chaîne sur *stdin*

Modifiez le programme afin que chaque lecture soit faite grâce à la fonction `litEtValideChaine`.

3. Exercice bonus : code refactoring

Cet exercice est relatif à l'exercice de gestion d'images (exercice 4) du TP4. Vous pouvez soit repartir de votre solution, soit repartir de la solution que nous avons mise à votre disposition sur Moodle.

Vous avez sans doute remarqué que le code source de la fonction *main* est assez long, voire très long. Nous vous demandons de réorganiser votre code de manière à le rendre plus lisible. Une manière d'y parvenir est d'introduire des fonctions faisant une partie du travail.

Concrètement, nous vous demandons de « factoriser » votre code en y introduisant au moins les fonctions suivantes :

- *afficherMenu* qui affiche le menu,
- *creerImage* (opérations 1 et 2) qui crée la matrice de pixels,
- *initPixels* (opération 1) qui initialise les pixels avec des valeurs aléatoires,
- *initPixels* (opération 2) qui initialise les pixels avec des valeurs prédéfinies,
- *afficherImage* (opération 3) qui affiche la valeur des pixels de l'image,
- *redimensionner* (opération 4) qui change la taille de l'image,
- *histogramme* (opération 5) qui renvoie l'histogramme de l'image,
- *afficherHistogramme* (opération 5) qui affiche l'histogramme de l'image,
- *libererImage* (opération 6) qui libère la mémoire dynamique de l'image et met celle-ci à NULL.

Testez votre programme avec *valgrind*.

4. Exercices bonus : spécifications

Ecrivez les specs des fonctions que vous avez conçues au point 3.