**Examen IN102 / CSC_3IN02_TA**
10 décembre 2024
Goran Frehse

# Énoncé

ENSTA ParisTech

**Informations générales :**
- Le sujet comporte **5** pages et l'examen dure **2** heures.
- Le barème est **volontairement** approximatif.

**Autorisations :**
- Les documents (polys, transparents, TDs, livres ...) sont autorisés.
- Sont **absolument interdits :** le web, le courrier électronique, les messageries, les répertoires des camarades, le téléphone (même pour avoir l'heure puisque vous l'avez sur votre ordinateur).

**Nommage et Formatage :**
- Dans les indications de format, le symbole ␣ représente un espace, ↵ représente un retour à la ligne et ⟶ indique que l'affichage souhaitée du programme suit.
- Vous devez respecter les règles de **nommage** des fichiers et des fonctions qui vous sont données.
- Le programme ne doit **afficher uniquement ce qui est explicitement demandé**, en respectant exactement le format (espaces, virgules, etc.).
- **En cas de non respect des consignes, la note peut être diminuée de 2 points**.

**Soumission :**
- **À la fin de l'examen**, vous devrez créer une **archive** contenant **tous les fichiers** que vous avez créés. Le nom de cette archive aura la structure suivante :
  `nom_prenom.zip`
  ou `.tgz` selon l'outil d'archivage que vous utilisez. Par exemple, Laura Croft nommera son archive `croft_laura.zip`. Avec l'outil zip, la commande serait
  `zip -r croft_laura.zip nom_du_dossier_a_zipper`
- Vous devrez **soumettre** cette archive sur le site
  `https://tinyurl.com/27bj8vv3`
  Par mesure de précaution, merci d'envoyer vos **fichiers sources (.c, .h) par mail** à votre chargé de TD, comme vous l'avez fait pour la validation des TDs).

# 1 Manipulation de chaînes ($\sim$ 15%)

Écrivez un programme qui prend en ligne de commande une chaîne de caractères (en minuscules, sans accents) et affiche une nouvelle chaîne où chaque voyelle (a, e, i, o, u et y) a été remplacée par le caractère *. Ensuite, sur une nouvelle ligne, le programme affichera le nombre de caractères remplacés.

**Attention :** La manipulation de la chaîne doit obligatoirement avoir lieu dans une fonction nommée `remplacer`. A l'intérieur de cette fonction, aucun affichage n'est permis. Récuperez la chaîne modifiée par `remplacer` dans la fonction `main` de votre programme, pour ensuite effectuer les affichages nécessaires.

**Nommage :** Le fichier source de ce programme devra s'appeler voyelles.c.

**Exemple de test :**

```
./voyelles.x␣programmation
pr*gr*mm*t**n
5
```

## 2 Vote préférentiel ($\sim 20\%$)

On considère un système de vote préférentiel permettant à $N$ électeurs de classer des candidats $1 \ldots M$ par ordre de préférence (le numéro 0 est réservé pour représenter un choix invalide). Dans ce système, chaque électeur exprime son **premier choix** et son **deuxième choix**. Ces votes sont enregistrés dans deux tableaux : $P$ pour le premier choix et $D$ pour le deuxième choix, donc un électeur $i$ exprime ses choix avec $P(i)$ et $D(i)$. Si un candidat obtient plus des votes que les autres candidats dans $P$, il gagne l'élection. En cas d'égalité dans $P$, le deuxième choix est consulté. En cas d'égalité aussi sur le deuxième choix, l'élection est invalide, ce qui est représenté par la valeur 0. Vous devez implémenter ce système de vote à l'aide des fonctions suivantes :

1. La fonction `compter(int P[], int N, int candidat)` retourne le nombre de votes exprimés dans le tableau $P$ pour un candidat donné.
2. La fonction `premier(int P[], int N, int M)` trouve le (ou un) candidat ayant le plus de votes dans $P$. Si plusieurs candidats ont ce nombre de votes, elle peut retourner n'importe lequel.
3. La fonction `premier_egal(int P[], int D[], int N, int M)` parcourt tous les candidats pour trouver celui avec le plus de votes en prenant en compte les égalités comme suivant : D'abord, elle utilise `premier` pour trouver le meilleur candidat $c$ selon $P$ et `compter` pour trouver son nombre de votes. Elle retient aussi le compte dans $D$ pour ce candidat. Ensuite, elle parcourt tous les candidats en appliquant le protocole suivant : Si elle rencontre un candidat qui a le même nombre de votes que $c$ dans $P$, et que son compte de votes dans $D$ est strictement supérieur, il devient le meilleur. Si les deux ont le même compte dans $P$ et dans $D$, la fonction donne 0 (choix invalide).
4. Testez la fonction `premier_egal` avec l'exemple suivant :

   ```
   int N = 10;
   int M = 4;
   int P[10] = {1, 2, 3, 1, 4, 2, 3, 1, 4, 2};
   int D[10] = {2, 3, 1, 3, 2, 1, 4, 4, 3, 1};
   ```

   et affichez le résultat.

## 3 Histogramme ($\sim 30\%$)

Écrivez un programme en C qui calcule et affiche l'histogramme d'un tableau d'entiers donné en entrée, avec les fréquences représentées par des étoiles (*). Les valeurs du tableau sont comprises entre 0 et 9, et sont passées en ligne de commande. Créez et utilisez les fonctions suivantes :

1. **Calcul des fréquences :**
   `void calculer_frequences(int argc, char* argv[], int frequences[])`
   Parcourt les arguments et remplit un tableau `frequences` de taille 10 pour compter le nombre d'occurrences de chaque chiffre.
2. **Affichage horizontal :**
   Fonction : `void afficher_horizontal(int frequences[])`
   Affiche chaque valeur (de 0 à 9) suivie par une ligne d'étoiles correspondant au nombre d'occurrences.
3. **Affichage vertical :**
   Fonction : `void afficher_vertical(int frequences[])`
   Description : Affiche les fréquences sous forme de colonnes d'étoiles. Chaque colonne représente une valeur entre 0 et 9 (affichée en-dessous), et contient le nombre d'étoiles correspondant à la fréquence, alignées en bas. Les colonnes sont séparées de deux espaces. Voir l'exemple ci-dessous pour le format exact d'affichage.
   **Attention :** N'affichez pas de lignes vides, c'est à dire la première ligne doit contenir au moins une étoile si le tableau n'est pas vide.

**Nommage :** Le fichier source de ce programme devra s'appeler `histogram.c`.

**Ex. tests** :

```
./histogram.x␣1␣3␣3␣7␣3␣1␣0
0:␣*
1:␣**
2:
3:␣***
4:
5:
6:
7:␣*
8:
9:

␣␣␣␣␣␣␣␣␣*
␣␣␣*␣␣␣␣␣*
*␣␣*␣␣␣␣␣*␣␣␣␣␣␣␣␣␣␣␣*
------------------------------
0␣␣1␣␣2␣␣3␣␣4␣␣5␣␣6␣␣7␣␣8␣␣9
```

## 4 Manipulation de struct ($\sim 35\%$)

Écrivez un programme en C qui gère des rectangles en utilisant des struct. Chaque rectangle est défini par les coordonnées de son coin supérieur gauche (x, y), sa largeur w, et sa hauteur h, tous exprimés en nombres **flottants**. Les données des rectangles sont passées via la ligne de commande dans l'ordre : `x1 y1 w1 h1 x2 y2 w2 h2 ... xn yn wn hn`

Le premier rectangle représente le canevas (une surface limite). Le programme doit effectuer les étapes suivantes :

1. Définir un type `struct Rectangle`, qui servira par la suite pour représenter tous les rectangles.
2. Implémenter une fonction `calculer_aire` qui calcule l'aire d'un rectangle.
3. Implémenter une fonction `intersection` qui calcule un nouveau rectangle représentant l'intersection de deux rectangles. Si l'intersection est vide, et retournera un rectangle avec largeur $-1$ et hauteur $0$, dont l'aire sera alors $0$.
4. Implémenter une fonction `creer_tableau_rectangles` pour stocker tous les rectangles passés en ligne de commande dans un tableau. La fonction retournera un pointeur vers ce nouveau tableau.
5. Implémenter une fonction `reduction` qui prend en entrée un tableau de rectangles, pour ensuite réduire tous les rectangles aux dimensions du canevas (le premier rectangle dans le tableau) en calculant leur intersection avec celui-ci. Il est obligatoire d'utiliser la fonction `intersection` pour effectuer le calcul. Le résultat (les rectangles réduits) est à stocker dans un *nouveau tableau* qui sera créé à l'intérieur de la fonction `reduction`. La fonction retournera un pointeur vers ce nouveau tableau.
6. Implémenter une fonction `affichage` pour les aires des rectangles avant et après cette réduction.
7. Appelez ces fonctions dans le `main` du programme pour lire les rectangles passés en ligne de commande, afficher les aires, reduire les rectangles au canevas et afficher les rectangles reduits. Consultez l'exemple ci-dessous pour le format précis de l'affichage. Libérez la mémoire des tableaux à la fin de leur utilisation.

**Astuce :** Vous pouvez utiliser la fonction `atof` pour convertir des chaînes en nombres flottants. Par exemple, `x = atof(chaine);`. Il faudra inclure `#include <stdlib.h>` pour l'utiliser.

**Nommage :** Le fichier source de ce programme devra s'appeler `rect.c`.

**Ex. tests :** Supposons des données suivantes :

- Canevas : (0, 0, 10, 10)
- Rectangle 1 : (2, 2, 5, 5)
- Rectangle 2 : (8, 8, 5, 5)
- Rectangle 3 : (12, 12, 3, 3)

Le programme serait exécuté ainsi :

```
./rect.x␣0␣0␣10␣10␣2␣2␣5␣5␣8␣8␣5␣5␣12␣12␣3␣3
Rectangle␣1␣:␣25.00
Rectangle␣2␣:␣25.00
Rectangle␣3␣:␣9.00
Rectangle␣1␣:␣25.00
Rectangle␣2␣:␣4.00
Rectangle␣3␣:␣0.00
```

**—— Fin du sujet ——**