IN 102 - Examen
25 octobre 2022
Goran Frehse

Énoncé

ENSTA ParisTech

Informations générales :
- Le sujet comporte **7** 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.
- Votre travail sera en partie évalué par un mécanisme automatique. 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**.

Commentaires :
- Si vos programmes contiennent des commentaires qui clarifient la structure et l'intention du code, **vous pouvez bénéficier d'un bonus de maximum 2 points**. Vous pouvez commenter même si le code est incomplet ou incorrect.

Soumission :
- **À la fin de l'examen**, vous devrez créer une **archive** contenant **tous** les fichiers **sources** que vous avez écrits (`.c`, `.h`). Le nom de cette archive devra avoir la structure suivante :
  `nom_prenom.zip` ou `.tgz` (selon l'outil d'archivage que vous utilisez). Par exemple, Laura Croft nommera son archive `laura_croft.zip`. Avec l'outil zip, la commande serait
  `zip -r laura_croft.zip nom_du_dossier_a_zipper`
- Vous devrez **copier** cette archive dans le répertoire de rendu se trouvant à
  `salle:/home/uei/frehse/in102rendus/`,
  en bien spécifiant votre **login**. Supposé que le login de Laura Croft est `croft`, la commande est
  `scp -o user=croft salle:/home/uei/frehse/in102rendus/ laura_croft.zip`
  Pour être sûr, merci d'également l'envoyer par mail à votre chargé de TD (comme pour la validation des TDs).

1 Intervalle d'une série ($\sim$ 15%)

On souhaite écrire un programme qui calcule l'intervalle d'une série de nombres flottants donnés en argument, c'est à dire la différence entre la plus grande valeur et la plus petite valeur. Par exemple, l'appel `./inter.x 12 -3 4.7 1` devra **afficher** 15, car $15 = 12 - (-3)$. On supposera qu'au moins un nombre est donné.

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

Format de sortie : Uniquement la valeur numérique de l'intervalle de la série **suivie d'un retour à la ligne**.

Exemples de test :

— `./inter.x 1234`↵
⟶
0↵

— `./inter.x 1234 1235`↵
⟶
1↵

— `./inter.x 1 2 3`↵
⟶
2↵

— `./inter.x 1.1 -2.2 3.3 -4.4`↵
⟶
7.7↵

2 Comptage de lettre ($\sim 20\%$)

Écrivez un programme qui prend en ligne de commande un argument représentant une chaîne de caractères (en un seul mot formé de lettres "a" à "z", sans espace, en *minuscules*). Ensuite, le programme affiche toutes les lettres du mots dans l'ordre alphabétique, avec pour chaque lettre le nombre de fois que cette lettre apparaît dans la chaîne.

Astuce : Créez un tableau pour compter chaque lettre de l'alphabet. Initialisez le tableau avec zero. Ensuite, parcourez le mot pour augmenter le compte correspondant à chaque lettre. Ensuite, affichez seulement les cases du tableau (avec leur lettre) ou le compte est plus grand que zero.

Rappel : Le code ASCII de 'a' est 97, celui de 'z' est 122.

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

Format de sortie : Chaque lettre, suivi par " :", le nombre d'occurrences, suivi par une virgule (pas de virgule à la fin). En cas d'argument absent, trop d'arguments ou de mauvais argument on affichera le message "Erreur. Usage : lettre.x mot".

Ex. tests :

- `./lettre.x test` ⟶ `e:1,s:1,t:2`↵
- `./lettre.x suisse` ⟶ `e:1,i:1,u:1,s:3`↵
- `./lettre.x deux mots` ⟶ Erreur.␣Usage␣:␣lettre.x␣mot.↵
- `./lettre.x avec trois mots` ⟶ Erreur.␣Usage␣:␣lettre.x␣mot.↵

3 Produit vectoriel ($\sim 25\%$)

Écrivez un programme dont l'objectif est de calculer le produit vectoriel de deux points 3D donnés en argument. Chaque point (x, y, z) est constituée par trois entiers, appelés x, y et z, et doit obligatoirement être représenté par un `struct`. Le produit vectoriel de deux points (x_1, y_1, z_1) et (x_2, y_2, z_2) est

$$(x_1, y_1, z_1) \wedge (x_2, y_2, z_2) = \begin{pmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{pmatrix}$$

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

Format de sortie : Le format d'affichage d'un point est (x, y, z) (sans espace). La sortie du programme **terminera par un retour à la ligne**.

Ex. tests : Le produit vectoriel $(1, 2, 3)$ et $(4, 5, 7)$ est calculé avec l'appel
— `./prod.x 1 2 3 4 5 7` $\longrightarrow (-1, 5, -3)$ ↵

Q1 Définissez dans votre programme un `struct` pour représenter un point. Dans votre fonction main, définissez deux points, a et b, à partir des arguments de la ligne de commande

Q2 Écrivez une fonction `void afficher` qui prend en argument une coordonnée et l'affiche (sans retour à la ligne).

Q3 Écrivez une fonction `void produit` qui prend en argument deux coordonnées c et d (ou des pointeurs) et **modifie** la coordonnée c pour qu'elle prenne les valeurs du produit vectoriel : $c \leftarrow c \wedge d$.
Attention : Cette fonction n'a pas de valeur de retour.

Q4 Dans la fonction `main` de votre programme, appelez la fonction `produit` pour calculer le produit de a et b et la stocker dans a. Ensuite, affichez a.

4 ASCII Mandelbrot ($\sim 40\%$)

Écrivez un programme pour calculer une image Mandelbrot, la stocker dans un tableau et l'afficher sous forme de caractères ASCII. La largeur w et l'hauteur h de l'image (en caractères) sera définie par les arguments de la ligne de commande. Procédez étape par étape, en suivant les indications données par les questions ci-dessous.

L'ensemble Mandelbrot est un ensemble de points (x, y) qui sont définis en utilisant la séquence suivante. En partant des valeurs $a = 0, b = 0, c = 0, k = 0$, on applique l'itération *Mandelbrot*

$$\begin{aligned} c &:= ab \\ a &:= a^2 - b^2 + x \\ b &:= 2c + y \\ k &:= k + 1 \end{aligned}$$

tant que $a^2 + b^2 \leqslant 4$ est satisfait (sinon, on s'arrête). L'ensemble Mandelbrot est constitué des points (x, y) pour lesquels $k \to \infty$. Pour des raisons pratiques, on s'arrête aussi quand k atteint la valeur $k_{max} = 100$. Chaque pixel de l'image correspond à une valeur (x, y). La "couleur" du pixel est la valeur k. Le coin en haut à gauche de l'image est à $(x_{\text{left}}, y_{\text{top}})$ et qui a une largeur de x_{width} et une hauteur de $y_{\text{height}} = x_{\text{width}} * h/w$. Utilisez les valeurs $x_{\text{left}} = -2, y_{\text{top}} = 1, x_{\text{width}} = 3$.

Vous allez calculer l'ensemble Mandelbrot sur une image de taille $w \times h$ qui représente les valeurs de sur un rectangle dont le coin en haut à gauche est à $(x_{\text{left}}, y_{\text{top}})$. Chaque "pixel" (i, j) est associé par interpolation linéaire à une coordonnée (x, y) où i est le numéro de ligne et j le numéro de colonne. On attribuera au pixel $(0, 0)$ la coordonnée $(x_{\text{left}}, y_{\text{top}})$ et au "pixel" $(w - 1, h - 1)$ la coordonnée $(x_{\text{left}} + x_{\text{width}}, y_{\text{top}} - y_{\text{height}})$.

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

Format de sortie : Dans l'image Mandelbrot, chaque caractère affiché sera un des 16 caractères suivants (le permier est un espace) : " `.,:;-+uco*#&8%@`"

Ex. tests :

```
./mandel.x␣30␣20
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣..@
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.:.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.@@,
␣␣␣␣␣␣␣␣␣␣␣␣␣.....@@..
␣␣␣␣␣␣␣␣␣␣␣␣␣.@:@@@@@.,,
␣␣␣␣␣␣␣␣␣␣␣␣␣..@@@@@@@@@.
␣␣␣␣␣␣␣.......@@@@@@@@@@.
␣␣␣␣␣␣␣.,.-..:@@@@@@@@@@,
␣␣␣␣␣␣␣.,@@@,@@@@@@@@@@@;
␣␣␣␣␣..,@@@@@@@@@@@@@@@@.
␣@@@@@@@@@@@@@@@@@@@@@@..
␣␣␣␣␣..,@@@@@@@@@@@@@@@@.
␣␣␣␣␣␣␣.,@@@,@@@@@@@@@@@;
␣␣␣␣␣␣␣.,.-..:@@@@@@@@@@,
␣␣␣␣␣␣␣.......@@@@@@@@@@.
␣␣␣␣␣␣␣␣␣␣␣␣␣..@@@@@@@@@.
␣␣␣␣␣␣␣␣␣␣␣␣␣.@:@@@@@.,,
␣␣␣␣␣␣␣␣␣␣␣␣␣.....@@..
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.@@,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.:.
```

```
./mandel.x␣15␣10
␣␣␣␣␣␣␣␣␣.@
␣␣␣␣␣␣␣␣␣@,
␣␣␣␣␣␣␣.:@@.,
␣␣␣␣...@@@@@.
␣␣␣␣,@,@@@@@;
␣@@@@@@@@@@.
␣␣␣␣,@,@@@@@;
␣␣␣␣...@@@@@.
␣␣␣␣␣␣␣.:@@.,
␣␣␣␣␣␣␣␣␣@,
```

Q1 Récupérez les premiers deux arguments de la ligne de commande dans des variables de type entier w et h. Par exemple, si on appelle le programme avec `./mandel.x 40 20`, vous stockez 40 dans w et 20 dans h.

Q2 Codez une fonction `compter` qui, donné des coordonnées (x, y) et $k_{\max}$, compte les nombre d'itérations k de Mandelbrot (jusqu'au maximum $k_{\max}$).
Astuce : Utilisez le type `double` pour les variables a, b, c.

Q3 Vous allez stocker l'image dans un tableau T de $w \times h$ entiers. Codez une fonction `calc_image` qui prend an argument $w, h, x_{\text{left}}, y_{\text{top}}, x_{\text{width}}, k_{\max}$ et qui retourne un pointeur vers un *nouveau tableau* qui contient l'image. N'oubliez pas que la mémoire occupée par ce tableau doit être libérée avant la fin du programme.

Pour chaque pixel (i, j) de l'image, appelez `compter` pour calculer la couleur et ensuite l'enregistrer dans `T[i*w+j]`.

Q4 Codez une fonction `afficher` qui prend en argument un tableau d'entiers, $w, h, k_{\max}$. Elle affiche les "pixels" de l'image sous formes de caractères ASCII on parcourant le tableau comme suivant :

1. Pour chaque "pixel", calculez une valeur entre 0 et 15 par interpolation linéaire de la couleur (valeur entre 0 et k_{max}). (Vous pouvez convertir un nombre flottant en entier en l'affectant à une variable entière.)
2. Affichez le caractère correspondant en utilisant le tableau C contenant les 16 caractères " `.,:;-+uco*#&8%@`" (les guillemets ne font pas partie du tableau ; le premier caractère est une espace).
3. A la fin de chaque "ligne" de l'image, affichez un retour à la ligne (`\n`) avant de passer à la ligne suivante.

Q5 Codez une fonction `histo` qui prend en argument un tableau d'entiers, $w, h, k_{\max}$ et affiche un *histogramme* de l'image : Comptez le nombre de "pixels" (caractères) pour chacune des "couleurs" entre 0 et 15. Affichez pour chacune des valeurs ce nombre, séparées par des virgules (pas de virgule à la fin).

Ex. tests :

```
./mandel.x␣30␣20
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣..@
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.:.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.@@,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣....@@..
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.@:@@@@@.,,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣..@@@@@@@@@.
␣␣␣␣␣␣␣␣.......@@@@@@@@@@.
␣␣␣␣␣␣␣␣.,.-..:@@@@@@@@@@,
␣␣␣␣␣␣␣␣.,@@@,@@@@@@@@@@;
␣␣␣␣␣␣..,@@@@@@@@@@@@@@@@@.
␣@@@@@@@@@@@@@@@@@@@@@@@@..
␣␣␣␣␣␣..,@@@@@@@@@@@@@@@@.
␣␣␣␣␣␣␣␣.,@@@,@@@@@@@@@@;
␣␣␣␣␣␣␣␣.,.-..:@@@@@@@@@@,
␣␣␣␣␣␣␣␣.......@@@@@@@@@@.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣..@@@@@@@@@.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.@:@@@@@.,,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣....@@..
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.@@,
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.:.
␣:415,.:48,,:8,::6,;:0,-:2,+:0,u:0,c:0,o:0,*:0,#:0,&:0,8:0,%:0,@:113
./mandel.x␣15␣10
␣␣␣␣␣␣␣␣␣.@
␣␣␣␣␣␣␣␣␣␣@,
␣␣␣␣␣␣␣␣.:@@.,
␣␣␣␣...@@@@@.
␣␣␣␣,@,@@@@@;
␣@@@@@@@@@@@.
␣␣␣␣,@,@@@@@;
␣␣␣␣...@@@@@.
␣␣␣␣␣␣␣␣.:@@.,
␣␣␣␣␣␣␣␣␣␣@,
␣:108,.:9,,:4,::2,;:0,-:0,+:0,u:0,c:0,o:0,*:0,#:0,&:0,8:0,%:0,@:27
```

—— Fin du sujet ——