Programmation impérative, projet 2022

Dates et principe

Cette page peut être mise à jour, avec informations complémentaires, précisions, *questions bonus*, etc. Pensez à y revenir souvent.

Projet à rendre pour le **8/1/2022** à **23h59**, aucun retard ne sera toléré.
Des soutenances pourront potentiellement être organisées ensuite.

Lire tout le sujet.

Un rendu de projet comprend :

- Un rapport typographié précisant vos choix, les problèmes techniques qui se posent et les solutions trouvées ; il présente en introduction le contexte et le sujet du projet, et il précise en conclusion les limites de votre programme. Le rapport sera de préférence composé avec LaTeX. Le soin apporté à la grammaire et à l'orthographe est largement pris en compte.
- Un code *abondamment* commenté ; la première partie des commentaires comportera systématiquement les lignes :
  1. `@ requires` décrivant les préconditions : c'est-à-dire conditions sur les paramètres pour une bonne utilisation (**pas de typage ici**),
  2. `@ assigns` listant les zones de mémoire modifiées,
  3. `@ ensures` décrivant la propriété vraie à la sortie de la fonction lorsque les préconditions sont respectées, le cas échéant avec mention des comportements en cas de succès et en cas d'échec,

  En outre chaque boucle `while` doit contenir un commentaire précisant la raison de sa terminaison (le cas échéant). De même en cas d'appels récursifs.
  On pourra préciser des informations additionnelles si des techniques particulières méritent d'être mentionnées.

  Le code doit enfin compiler sans erreur (évidemment) et sans warning lorsque l'option `-Wall` est utilisée. **Un code qui ne compile pas se verra attribuer la note de 0.**

- Un manuel d'utilisation de votre exécutable, même minimal, est toujours bienvenu.

Avez-vous lu tout le sujet ?

Protocole de dépôt

Vous devez rendre

- votre rapport (en pdf) et
- vos fichiers de code

rassemblés dans une archive tar gzippée identifiée comme *votre_prénom_votre_nom*`.tgz`.
La commande devrait ressembler à :
`tar zcvf randolph_carter.tgz rapport.pdf fichiers.c autres_trucs_éventuels.c...`
N'OUBLIEZ surtout PAS de mettre le nom identifiant l'archive (donc nouveau) **en PREMIER.**
Lisez le man ! et testez le contenu de votre archive (une commande comme par exemple :
`tar tvf randolph_carter.tgz` doit lister les fichiers et donner leur taille).

- **Une archive qui ne contient pas les fichiers demandés ne sera pas excusable.**

- ****Une archive qui n'est pas au format demandé** (archive tar gzippée avec suffixe .tgz) **ne sera pas corrigée**** donc c'est 0/20.

Toute tentative de fraude (plagiat, etc.) sera sanctionnée. Si plusieurs projets ont **des sources trop similaires** (y compris sur une partie du code uniquement), *tous* leurs auteurs se verront attribuer la note 0/20. En particulier, il faudra prendre soin de ne pas publier son travail sur un dépôt public (en tout cas pas avant la date de fin de rendu). On évitera également de demander (ou de donner) des conseils trop précis à ses camarades (y compris des promotions précédentes), ces conseils ayant pu être donnés à plusieurs personnes. Les rendus seront comparés deux à deux.

Procédure de dépôt
Vous devez enregistrer votre archive tgz dans le dépôt dédié au cours PRIM11 (prim11-2022) en vous connectant à http://exam.ensiie.fr. Ce dépôt sera ouvert jusqu'au 8 janvier inclus.

Contexte

Le but de ce projet est d'implémenter un interpréteur pour un petit langage graphique.

Interface

L'exécutable final devra lire sur l'entrée standard un fichier qui contient la taille de l'image à calculer, puis une série de caractères qui indiqueront comment construire l'image. L'image calculée sera ensuite écrite sur la sortie standard sous la forme d'un fichier PPM.

Si le programme final s'appelle `prog`, et qu'on a un fichier d'entrée nommé `test.ipi`, il devrait être possible d'entrer la commande

```
./prog < test.ipi | display
```

pour calculer l'image décrite dans `test.ipi` et l'afficher en utilisant la visionneuse d'Image Magick.

État de la machine

Le calcul de l'image à produire se fait à l'aide d'une machine à états. On dispose de jusqu'à dix calques surposés, chacun étant de la taille de l'image à produire. On peut déplacer un curseur sur le calque du dessus, et tracer des lignes entre la position courante et la dernière position marquée. On peut aussi remplir la couleur à la position du curseur, comme avec l'outil « remplissage » des outils d'imagerie courants. La couleur à utiliser est déterminée à l'aide d'un seau dans lequel on peut ajouter des doses de couleurs de bases ; la couleur courante est alors la moyenne des couleurs contenues dans le seau. Un autre seau contiendra des doses d'opacité, ce qui permettra de calculer l'opacité courante.
Un état de cette machine sera constitué des éléments suivants :

- deux entiers indiquant la position courante du curseur ;
- deux entiers indiquant la dernière position marquée ;
- un élément qui indique la direction du curseur ; celle-ci peut être Nord, Est, Sud ou Ouest ;
- un multiensemble de couleurs qui représente le seau dans lequel sont mélangées des doses de couleurs pour produire la couleur courante ;
- un multiensemble d'opacités qui représente le seau dans lequel sont mélangées des doses d'opacités pour produire l'opacité courante ;
- une pile de calques de taille au plus 10, chaque calque étant une grille de pixels de la taille de l'image à produire.

On aura besoin des types de données suivants :

- Une composante est un entier non signé sur 1 octet.
- Une couleur est formée de trois composantes pour le rouge, le vert et le bleu.

- Une opacité est donnée par une composante ; 0 correspond à la transparence totale, tandis que 255 correspond à l'opacité totale.
- Un pixel est formé par une couleur et une opacité.
- Un calque est une grille carrée de pixels, de la taille de l'image à produire au final.

Initialement, l'état de la machine est le suivant :

- les positions courante et marquée sont toutes les deux en (0,0) ;
- la direction est Est ;
- les seaux de couleurs et d'opacités sont vides tous les deux ;
- la pile contient un seul calque ; chaque pixel de ce calque est formé de la couleur (0,0,0) et de l'opacité 0.

Calcul du pixel courant

Le pixel courant correspondant au contenu des seaux est calculé de la façon suivante :

- on fait la moyenne (entière) des opacités du seau d'opacités, ce qui nous donne l'opacité courante ;
- si ce seau est vide, on prend 255 comme valeur pour l'opacité courante ;
- on fait la moyenne (entière) de chacune des composantes des couleurs contenues dans le seau de couleurs ;
- la valeur de chaque composante de la couleur du pixel courant est alors cette moyenne multipliée par l'opacité calculée précédemment puis divisée par 255 ;
- si le seau de couleurs est vide, on prend comme valeur 0 pour chacune des composantes de la couleur courante.

Par exemple, si le seau de couleurs contient deux doses de rouge (255,0,0) et une dose de jaune (255,255,0), et que le seau d'opacités contient une dose d'opacité complète (255) et une dose de transparence totale (0), la moyenne des doses d'opacité sera de 127. La moyenne des composantes des doses de couleur sera (255,85,0). Le pixel courant aura donc pour couleur (127, 42, 0) et pour opacité 127.

Format d'entrée

Le format proposé est composé d'une première ligne, qui contient un entier donné en mode texte. Cet entier correspond à la largeur et la hauteur de l'image qui sera produite. (Celle-ci est donc carrée.)

La suite du fichier contient des caractères dont certains modifieront l'état de la machine :

`n`
: Ajoute la couleur noire (c'est-à-dire (0,0,0)) dans le seau de couleurs.

`r`
: Ajoute la couleur rouge (c'est-à-dire (255,0,0)) dans le seau de couleurs.

`g`
: Ajoute la couleur verte (c'est-à-dire (0,255,0)) dans le seau de couleurs.

`b`
: Ajoute la couleur bleue (c'est-à-dire (0,0,255)) dans le seau de couleurs.

`y`
: Ajoute la couleur jaune (c'est-à-dire (255,255,0)) dans le seau de couleurs.

`m`
: Ajoute la couleur magenta (c'est-à-dire (255,0,255)) dans le seau de couleurs.

`c`
: Ajoute la couleur cyan (c'est-à-dire (0,255,255)) dans le seau de couleurs.

`w`
: Ajoute la couleur blanche (c'est-à-dire (255,255,255)) dans le seau de couleurs.

`t`
: Ajoute l'opacité transparente (0) dans le seau d'opacités.

`o`
: Ajoute l'opacité complète (255) dans le seau d'opacités.

`i`
: Vide les seaux de couleurs et d'opacité.

`v`
: Avance la position du curseur d'un pas dans la direction courante ; si le bord de l'image est atteint, on repasse de l'autre coté.

`h`
: Tourne la direction courante dans le sens horaire.

`a`
: Tourne la direction courante dans le sens anti-horaire.

`p`
: Met à jour la position marquée en y mettant la position actuelle du curseur.

`l`
: Trace une ligne entre la position du curseur et la position marquée, en utilisant l'algorithme décrit ci-dessous.

`f`
: Remplit la zone de même couleur autour du curseur, en la remplaçant par la couleur courante, à l'aide de l'algorithme décrit ci-dessous.

`s`
: Ajoute un nouveau calque dans la pile de calques ; chaque pixel de ce nouveau calque sera formé de la couleur (0,0,0) et de l'opacité 0 ; s'il y a déjà 10 calques dans la pile, ne fait rien.

`e`
: Fusionne les deux calques les plus hauts dans la pile, comme décrit ci-dessous ; il y aura donc un calque de moins dans la pile ; s'il n'y a qu'un seul calque, ne fait rien.

`j`
: Découpe le calque situé juste en dessous du sommet de la pile en utilisant comme masque les opacités de celui au sommet (cf. ci-dessous) ; retire celui au sommet ; s'il n'y a qu'un seul calque, ne fait rien.

Les autres caractères seront ignorés, y compris les versions en majuscule des caractères ci-dessus.

Tracé de lignes

Pour tracer une ligne entre la position (x0, y0) à la position (x1, y1), on utilisera l'algorithme suivant : (`|x|` représente la valeur absolue de `x`, toutes les opérations se font sur les entiers)

```
δx := x1 - x0
δy := y1 - y0
d := max(|δx|, |δy|)
si δx et δx sont non nuls et de même signe
alors s := 0
sinon s := 1
x := x0 * d + (d - s) / 2
y := y0 * d + (d - s) / 2
répéter d fois
  changer le pixel en (x/d, y/d) par le pixel courant
  x := x + δx
  y := y + δy
changer le pixel en (x1, y1) par le pixel courant
```

Le pixel courant est celui correspondant aux seaux, cf. supra.

Remplissage

Pour remplir une zone de couleur depuis la position (x,y) sur le calque `calque`, on appellera la fonction `fill` décrite ci-dessous avec comme paramètres `x, y, calque[x,y], courant, calque` où `courant` est le pixel courant, à condition que ce pixel courant soit différent de celui déjà en (x,y), sinon on ne fait rien.

```
fonction fill(x, y, ancien, nouveau, calque)
  si calque[x,y] = ancien
  alors
    calque[x,y] := nouveau
    si x > 0 alors fill(x-1, y, ancien, nouveau, calque)
    si y > 0 alors fill(x, y-1, ancien, nouveau, calque)
    si x < size-1 alors fill(x+1, y, ancien, nouveau, calque)
    si y < size-1 alors fill(x, y+1, ancien, nouveau, calque)
```

où `size` est la taille de l'image à produire.

Remarque : sur de grands calques, une implémentation récursive de la fonction `fill` comme décrit ci-dessus peut conduire à des dépassements de pile. Vous devrez plutôt utiliser une pile explicite dans laquelle vous mettrez l'ensemble des positions qu'il reste encore à traiter.

Fusion de calques

Pour fusionner deux calques c0 et c1, on procède de la manière suivante pour chacun des pixels du nouveau calque :

- on note α0 l'opacité de c0 à cette position ;
- la valeur de chacune des composantes du nouveau pixel (couleur et opacité) vaut celle en c0 plus (celle en c1 fois (255 - α0) divisé par 255).

Par exemple, si le pixel de c0 a pour couleur (127, 42, 0) et pour opacité 127, et que celui en c1 a pour couleur (0, 100, 200) et pour opacité 200, le pixel de la fusion aura pour couleur (127, 92, 100) et pour opacité 227.

Remarque : il n'y a pas de dépassement de capacité pour les nouvelles composantes, parce que de la façon dont est calculé le pixel courant, les composantes des couleurs ne peuvent pas être plus grandes que la composante de l'opacité.

Découpage de calques

Pour découper un calque c1 en utilisant l'opacité d'un calque c0 comme masque, on procède de la manière suivante pour chacun des pixels du nouveau calque :

- on note α0 l'opacité de c0 à cette position ;
- la valeur de chacune des composantes du nouveau pixel (couleur et opacité) vaut celle en c1 fois α0 divisé par 255.

Par exemple, si le pixel de c0 a pour couleur (127, 42, 0) et pour opacité 127, et que celui en c1 a pour couleur (0, 100, 200) et pour opacité 200, le pixel de la fusion aura pour couleur (0, 49, 99) et pour opacité 99.

Production de l'image finale

Une fois l'entrée complètement lue, on prendra le calque situé en haut de la pile pour obtenir l'image finale. On ne prendra en compte que les couleurs et on oubliera donc l'opacité.

On écrira sur la sortie standard l'image correspondante au format PPM vu en cours.

Exemples

On trouvera dans le fichier exemples_ipi_2022.tgz un certains nombres de fichiers d'entrées (*exemple*`.ipi`) avec l'image devant être produite (*exemple*`.ppm`), afin que vous puissiez tester votre programme.

D'autres exemples seront possiblement proposés d'ici la date de rendu.

Lors de la correction, votre programme sera testé avec des exemples qui n'auront pas été fournis à l'avance ; faire passer les exemples ne suffira donc pas, il faudra bien respecter la spécification.

Bonus

- Permettre au programme de prendre des arguments en ligne de commande :

    - si un premier argument est fourni, ce sera le nom d'un fichier qui sera lu en entrée à la place de l'entrée standard ;
    - si un deuxième argument est fourni, ce sera le nom d'un fichier qui sera écrit en sortie à la place de la sortie standard ;
    - les autres arguments seront ignorés, avec un avertissement sur la sortie d'erreur standard.
- Ajouter une interface graphique qui montre la construction de l'image au fur et à mesure de son calcul.

Conseils

Pour la récupération d'une entrée de l'utilisateur, plutôt que faire un `scanf` directement, il vaut parfois mieux récupérer une ligne en entier avec `fgets` puis utiliser `sscanf` dessus ; on peut utiliser la suite de commandes suivantes :

```
char buf[256];
 .
 .
 .
fgets(buf, 256, stdin);
sscanf(buf, "format", ...);
```

Pour lire un seul caractère, on pourra utiliser `getc`, qui renvoie `EOF` en cas d'erreur ou en fin de fichier.

Il pourra être opportun d'utiliser certaines fonctions de la bibliothèque standard comme `memcpy` ou `memset`. Se reporter aux pages de manuel pour leur fonctionnement.

Vous devez avoir lu jusqu'ici avant de commencer.