Université Paris Diderot
Langage C (LC4) - 26 mai 2013 - Durée : 2h45
Aucun document autorisé. Ordinateurs et téléphones interdits / éteints

---

Il est fortement conseillé de lire le sujet dans son intégralité avant de commencer : il est possible de répondre à des questions sans avoir traité les précédentes, l'annexe contient des informations utiles, ...

Quand on demande d'écrire une séquence de code, il faut imaginer qu'elle sera intégrée dans un code plus général. La réponse contiendra alors deux parties : la première sera constituée de déclarations et de définitions et la seconde d'une suite d'instructions.

Présentation générale

Ce sujet suit les grandes lignes d'un programme de correction automatique de QCM et en constitue une version très simplifiée.
L'objectif est donc d'analyser une feuille de réponses : une telle feuille est constituée d'un ensemble de cases carrées ne contenant à l'origine rien d'autre qu'un chiffre correspondant à un numéro de réponse (voir l'annexe en fin de sujet). Il va s'agir d'obtenir à partir de cette feuille la liste des réponses fournies à chacune des questions tout d'abord en mémoire, puis finalement dans un fichier sur le disque (on considérera ici qu'une réponse a été sélectionnée si le carré contient au moins 50% de points noirs).
On va travailler sur l'image obtenue en scannant une feuille, c'est-à-dire à partir d'un fichier dans un format utilisable au moyen d'une bibliothèque de fonctions : le choix a été fait d'utiliser le format `tiff` pour laquelle une bibliothèque est disponible [`libtiff.a`].
Au travers de cette bibliothèque, le programme associera à l'image scannée d'une feuille de réponses, une matrice de nombres entiers non signés de 32 bits : chaque nombre représente la couleur du pixel correspondant (voir l'annexe).
Par ailleurs, un fichier de configuration contient les informations permettant la localisation sur la feuille des différents carrés utilisés pour répondre (voir l'annexe).

1. Récupération et exploitation des données du fichier de configuration

1.1. Donner la définition d'un type `configuration` permettant de regrouper en mémoire et d'y accéder facilement les informations contenues dans le fichier de configuration.

1.2. Donner le prototype d'une fonction permettant de récupérer en mémoire les données contenues dans

2. Analyse de l'image

On trouvera dans l'annexe les informations extraites du fichier `tiffio.h` d'interface de la bibliothèque `libtiff.a` : elles concernent des types et constantes et des prototypes de fonctions qu'on utilisera pour traiter la suite.

2.1. Dans une première approche (optimiste), on suppose que toutes les images qu'on va traiter ont la même taille : elles ont toutes la même hauteur `HAUTEUR` et la même largeur `LARGEUR` (ces deux constantes sont supposées prédéfinies).

2.1.1. Définir au moyen de `typedef` le type `IMAGE1` correspondant au type *pointeur vers tableau à deux dimensions de* `HAUTEUR` *tableaux de* et `LARGEUR` colonnes *d'entiers non signés.*

2.1.2. Donner le code nécessaire (toutes les déclarations et définitions, appels de fonctions ....) permettant de récupérer en mémoire le tableau de `HAUTEUR` lignes et `LARGEUR` colonnes d'entiers non signés associé à une image au format tiff dans le fichier `feuille.tif` (voir l'annexe pour les fonctions disponibles).

2.1.3. Donner une expression donnant la valeur du pixel ayant pour coordonnées des valeurs `hauteur` et `largeur` données (cette valeur sera extraite de la zone mémoire initialisée en 2.1.2.

2.1.4. Quelles seraient les conséquences de l'utilisation des codes de 2.1.2 et 2.1.3 pour une image n'ayant pas les dimensions attendues ?
Donner une séquence permettant de vérifier que les dimensions de l'image sont bien `HAUTEUR` et `LARGEUR`.

2.2. L'expérience montre malheureusement que la taille de l'image scannée est variable : on va donc écrire du code non basé sur cette hypothèse.
Par alleurs, les dimensions n'étant pas contantes et pour respecter la norme ANSI, ce programme n'utilisera pas de tableau mais une zone mémoire repérée par un pointeur de type `unsigned int *` correctement initialisé.

2.2.1. Ecrire une séquence contenant toutes les définitions et instructions permettant de récupérer dans une telle zone mémoire la représentation de l'image produite par la fonction `TIFFImageRead`.

2.2.2. Ecrire le code d'une fonction qui renvoie la valeur d'un pixel identifié par ses coordonnées : cette valeur sera retrouvée dans la zone mémoire initialisée par l'appel à `TIFFImageRead` en 2.2.1.

2.2.3. Ecrire une fonction qui renvoie le nombre de pixels noirs dans le carré correspondant à une réponse donnée pour une question donnée.

2.2.4. Ecrire un programme qui
- analyse complètement une feuille de réponses et enregistre les résultats de cette analyse dans un tableau de nom `analyse` ayant autant de lignes qu'il y a de questions, chaque ligne ayant 10 colonnes. La valeur de `analyse[q][r]` sera `1` ou `0` selon que le carré correspondant à la réponse r de la question q contient plus de `50%` de points noirs.
- sauvegarde cette analyse sur disque : le fichier contiendra autant de lignes qu'il y a de questions et chaque ligne consistera en la suite des numéros des réponses choisies (il peut y avoir des lignes vides si aucune case n'a été cochée pour la question correspondante).

Ce programme utilisera les fonctions définies précédemment (vous pouvez utiliser les fonctions que vous n'avez pas écrites si vous en avez spécifié le prototype et l'effet).

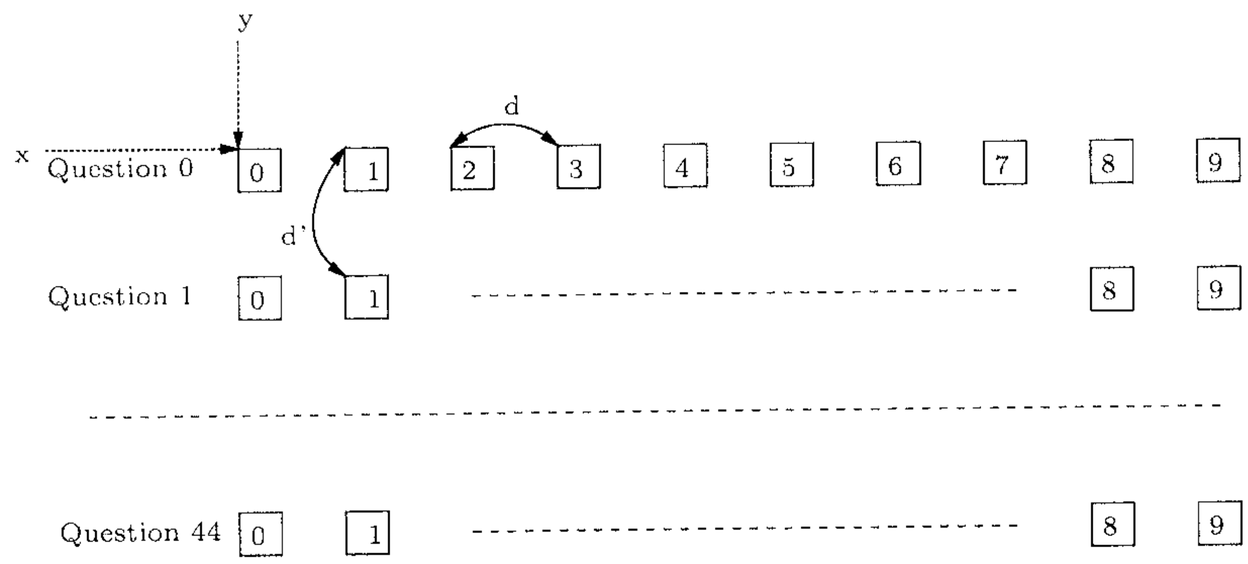
2.2.5. En supposant que
- le programme principal (2.2.4.) est dans le fichier `prog.c` ;
- toutes les fonctions que vous avez écrites sont dans un fichier `utilitaires.c` ;
- le fichier d'interface avec ces fonctions est `utilitaires.h` ;
- la bibliothèque utilisée s'appelle `libtiff.a` et elle est placée dans un répertoire standard des bibliothèques (typiquement `/usr/local/lib`)

donner un fichier `makefile` pour construire le binaire exécutable `prog` correspondant.

ANNEXE

1) L'organisation de la feuille de réponses

Il y a 45 questions (numérotées de 0 à 44) au maximum et pour chacune 10 choix de réponses (numérotés de 0 à 9) avec la disposition suivante :



2) Les données fournies dans un fichier de configuration

Le fichier de configuration d'un questionnaire permet de définir la struture d'une feuille de réponse.
Il consiste en une ligne contenant les nombres suivants (en codage ASCII) séparés par des espaces :

- le nombre effectif (≤ 45) de questions d'un questionnaire ;
- la position en hauteur (repérée x sur la figure), exprimée en nombre de pixels, du coin supérieur gauche du carré de la première ligne contenant le chiffre 0 ;
- la position en largeur (repérée y sur la figure), exprimée en nombre de pixels du coin supérieur gauche du carré de la première ligne contenant le chiffre 0 ;
- la longueur du côté des carrés, exprimée en nombre de pixels ;
- la distance, exprimée en nombre de pixels (repérée d' sur la figure) entre les coins supérieurs gauche de deux carrés successifs d'une même colonne (même chiffre dans les carrés) ;
- la distance, exprimée en nombre de pixels (repérée d sur la figure) entre les coins supérieurs gauche de deux carrés successifs d'une même ligne (chiffres successifs dans les carrés).

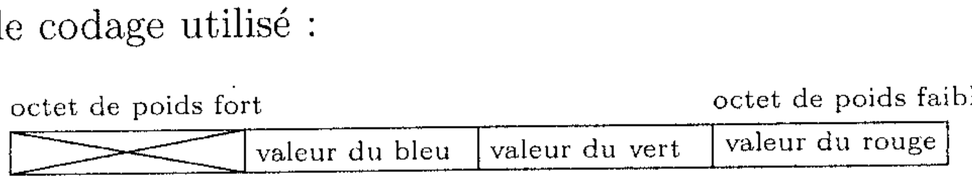
Un exemple de contenu possible pour le fichier est

```
24 700 450 50 60 100
```

qui spécifie qu'il y a 24 questions dans le questionnaire, que le coin supérieur gauche du premier carré (carré contenant 0) est en (700, 450), que le côté de chaque carré contient 50 pixels et que la distance entre deux coins supérieurs gauches est en hauteur de 60 pixels et en largeur de 100 pixels.

3) Le codage RGB (en français RVB pour Rouge/Vert/bleu des couleurs

Une couleur est obtenue par composition de 3 couleurs primitives (rouge, vert et bleu) plus d'autres informations que nous n'utiliserons pas ici. Chaque couleur se caractérise par sa luminosité dans ces couleurs de base. Cette luminosité a une valeur comprise entre 0 et 255 en décimal (entre 0x0 et 0xff en hexadécimal).
Le schéma suivant résume le codage utilisé :



Rappelons enfin que le noir (non-couleur) correspond à la valeur 0 pour chacune des trois couleurs de base et le blanc à la valeur 255 pour chacune de ces couleurs.

4) Informations relatives à la bibliothèque `libtiff.a`

[redacted]

- la fonction `TIFF *TIFFOpen(char *reference, char *mode)` où les paramètres sont analogues à ceux de `fopen` ouvre l'image dans le fichier de référence spécifiée dans le mode demandé (ici il s'agira de `"r"`). La fonction renvoie `NULL` en cas de problème et sinon, un pointeur de type `TIFF *` permettant d'accéder au fichier ;
- la fonction `int TIFFGetField(TIFF *, unsigned int, unsigned int *)` permet de récupérer les attributs de l'image.
  Le second paramètre permet de sélectionner l'attribut et le troisième de récupérer la valeur correspondante : les constantes symboliques `TIFFTAG_IMAGEWIDTH` et `TIFFTAG_IMAGELENGTH` permettent respectivement de sélectionner la largeur et la hauteur de l'image.
  Les valeurs renvoyées sont des nombres de pixels ;
- la fonction `int TIFFImageRead(TIFF *tiff, int hauteur, int largeur, void *adr)` copie en mémoire à l'adresse `adr` l'image associée au descripteur `tiff` de hauteur et largeur spécifiées (cette image est une séquence de `hauteur*largeur` entiers non signés). La séquence contient successivement les pixels de la premiére ligne, puis ceux de la seconde, ... La fonction renvoie la valeur `0` en cas de problème.

5) Rappels sur les fonctions de la bibliothèque standard

le type `FILE` pour manipuler les fichiers dans un programme ;

- la fonction `FILE *fopen(char *reference, char *mode)` pour acquérir un `FILE *` ;
- la fonction `int fscanf(FILE *fichier, char *format, ...)` se comporte comme `scanf` mais lit les données dans le fichier et non sur l'entrée standard (le terminal) ;
- la fonction `int fprintf(FILE *fichier, char *format, ...)` se comporte comme `printf` mais écrit les valeurs dans le fichier et non sur la sortie standard (le terminal).