TP 1 : tableaux et pointeurs

Programmation en C (LC4)

Semaine du 29 janvier 2007

1 Manipulation de la ligne de commande

1.1 Fonctions utiles

Nous aurons souvent besoin de la fonction fort utile « `printf(const char* format, arg1, ...)` ». Pour utiliser cette fonction, il faut ajouter au début du fichier « `#include <stdio.h>` ». Voilà un exemple d'utilisation de « `printf()` » :

```
int a = 10; char b='z';
printf("valeurs de a : %i, b : %c, argv[0] : %s \n",a,b,argv[0]);
```

affiche à l'exécution du programme, dans votre terminal :
« `valeurs de a : 10, b : z, argv[0] : ./mon_programme` »
Pour ce TP, nous aurons aussi besoin d'une fonction standard permettant la comparaison des chaînes de caractères. Pour l'utiliser, vous devez rajouter « `#include <string.h>` » au début de votre fichier. La fonction « `int strcmp(char * cs, char * ct)` » compare la chaîne `cs` à la chaîne `ct` et renvoie une valeur négative si `cs<ct` (ordre lexicographique[1]), nulle si `cs==ct` et positive si `cs>ct`.

1.2 Exercices

Les fonctions suivantes travaillent sur un tableau de chaînes de caractères « `char *tab[]` ». Vous pourrez par exemple les tester sur la ligne de commande du programme, passée en argument à la fonction `main` du programme.

Exercice 1 Écrire une fonction « `void affichage(int n, char *tab[])` » qui écrit dans le terminal les chaînes de caractères du tableau `tab`, séparées par des espaces.

Exercice 2 Écrire une fonction « `void echange_mot(int i, int j, char *tab[])` » qui échange les $i^{ème}$ et $j^{ème}$ mots dans le tableau `tab`.

Exercice 3 Écrire une fonction « `void echange_lettre(int i, int j, char *tab[])` » qui échange la première lettre du $i^{ème}$ mot avec celle du $j^{ème}$.

Exercice 4 — *bonus* En vous aidant des fonctions vues en TD, implémentez une fonction « `tri_permutation(int n, int perm[], char *tab[])` » qui trie un tableau de chaînes de caractères, pour l'ordre lexicographique.

2 Des images comme des tableaux

2.1 Représentation des images

On représentera une image par un tableau d'entiers à une dimension, chaque entier représentant la couleur d'un pixel. Ici, une couleur sera juste un entier entre 0 (noir) et 255 (blanc), représentant un niveau de gris. Les pixels sont donnés dans l'ordre suivant :

[1] Remarque : pour comparer deux chaînes dans l'ordre lexicographique, on se place à la fin de leur plus long préfixe commun (i.e. au premier indice où elles diffèrent), et on les compare par le caractère à cette position. Par exemple : `"abcd"` est plus petit que `"abzd"` puisque `'c'` est plus petit que `'z'`.

– le premier pixel est le coin supérieur gauche
– suivent les pixels de sa ligne, de gauche à droite
– suit la deuxième ligne, toujours de gauche à droite
– et ainsi de suite jusqu'en bas
Toutes les fonctions à écrire seront du type :

void ma_fonction(**int** width, **int** height,**int** image[])

où `width` est la largeur en pixels, `height` la hauteur en pixels, et `image` le tableau représentant l'image comme décrit ci-dessus.
Le pixel de coordonnées (x, y) (en prenant l'origine au coin supérieur gauche de l'image, et en orientant l'axe des ordonnées vers le bas) s'obtiendra donc par `image[x+width*y]`.

2.2 Exercices

Exercice 5 Écrivez une fonction « `void miroir_vertical(int width, int height, int image[])` » qui inverse l'image suivant un axe de symétrie vertical.

Exercice 6 Écrivez une fonction « `void miroir_horizontal(int width,int height,int image[])` » suivant un axe de symétrie horizontal.

Exercice 7 Écrivez une fonction « `void rend_flou(int width,int height,int imagesrc[],int imagedst[])` » qui écrit dans `imagedest` une version floutée de `imagesrc`. L'idée est de remplacer chaque pixel par une moyenne (éventuellement pondérée) entre lui et ses voisins.

2.3 Pour tester vos fonctions

Vous pouvez récupérer les fichiers `ppm.c`, `ppm.h` et `einstein.ppm` disponibles à l'adresse

`http://www.di.ens.fr/~pmaurel/LC4/tp1/`

Le fichier `ppm.c` fournit les deux fonctions suivantes :

void charge_image_ppm (**const char*** filename, **int** width, **int** height, **int** image[]);
void enregistre_image_ppm (**const char*** filename, **int** width, **int** height, **const int** image[])

qui permettent de charger une image depuis un fichier ou d'enregistrer une image dans un fichier.
`einstein.ppm` est une image. Sa taille est 600x782 pixels.
Votre fonction `main` devrait ressembler à :

```
int main(int argc,char *argv[]) {
  int image[600*782];
  int width=600;
  int height=782;
  charge_image_ppm(argv[1],width,height,image);
  miroir_horizontal(width,height,image);
  enregistre_image_ppm(argv[2],width,height,image);
  return(0);
}
```

On lance le programme avec en premier argument le nom du fichier contenant l'image sur laquelle on veut travailler (a priori `einstein.ppm`), et en deuxième argument le nom du fichier dans lequel on veut enregistrer le résultat. Par exemple, `./a.out einstein.ppm einsteinmiroir.ppm`.
Il faudra avoir au préalable compilé `ppm.c` avec la commande `gcc -c ppm.c` (à faire une seule fois), puis compiler votre `.c` (mettons que vous l'avez appelé `image.c`) avec `gcc ppm.o image.c`.