TP 1 : tableaux et pointeurs

Programmation en C (LC4)

Semaine du 28 janvier 2008

1 Manipulation de la ligne de commande

1.1 Fonctions utiles

Nous aurons souvent besoin de la fonction « `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` »

Attention, il faut respecter les types : `%i` ou `%d` pour des entiers, `%c` pour des caractères, `%s` pour des char*.

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**

```
void affichage(int n, char** tab)
{
  int i;
  for(i=0;i<n;i++)
  {
    printf("%s ",tab[i]);
  }
  printf("\n");
}
```

► **Exercice 2**

[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'`.

```
void echange_mot(int i, int j, char** tab)
{
  char* tmp=tab[i];
  tab[i]=tab[j];
  tab[j]=tmp;
}
```

► **Exercice 3**

```
void echange_lettre(int i, int j, char** tab)
{
  char tmp=tab[i][0];
  tab[i][0]=tab[j][0];
  tab[j][0]=tmp;
}
```

► **Exercice 4**

```
void tri_permutation (int n, int perm[], char ** tab,char ** res) {
  int i = 0;
  int j = 0;
  for (i = 0; i < n; i++)
    perm[i] = 0;
  for (i = 0; i < n; i++)
    for (j = i + 1; j < n; j++)
      if (strcmp(tab[i], tab[j]) > 0)
        perm[i]++;
      else
        perm[j]++;

  for (i = 0; i < n; i++)
  {
    res[perm[i]]=tab[i];
  }

}
```

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 :
– 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**

```
void miroir_vertical(int width, int height, int image[]){
int x,y,tmp;
for(y=0;y<height;y++){
    for(x=0;x<width/2;x++){
tmp=image[x+width*y];
image[x+width*y]=image[(width-1-x)+width*y];
image[(width-1-x)+width*y]=tmp;
}
  }
  }
```

► **Exercice 6**

```
void miroir_horizontal(int width, int height, int image[]){
int x,y,tmp;
for(y=0;y<height/2;y++){
    for(x=0;x<width;x++){
tmp=image[x+width*y];
image[x+width*y]=image[x+width*(height-1-y)];
image[x+width*(height-1-y)]=tmp;
}
}
}
```

► **Exercice 7**

```
// calcule la moyenne des 8 points de img[]
// qui sont autour du point (x,y) et de (x,y)
int moyenne(int w, int h, int x, int y, int img[]){
  return (( img[(x-1)+w*(y-1)] + img[(x-1)+w*y] + img[(x-1)+w*(y+1)]+
            img[x+w*(y-1)] + img[x+w*y]+ img[x+w*(y+1)] +
            img[(x+1)+w*(y-1)] + img[(x+1)+w*y] + img[(x+1)+w*(y+1)] )
          /9 );
}


void rend_flou(int w, int h, int src[], int dst[]){
  int x,y;
  for(y=0;y<h;y++){
    for(x=0;x<w;x++){
      if(y==0 || x==0 || y==(h-1) || x==(w-1)){
        dst[x+w*y]=src[x+w*y];
      }
      else{
        dst[x+w*y]=moyenne(w,h,x,y,src);
      }
    }
  }
}
```

2.3 Pour tester vos fonctions

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

`http://cedric.cnam.fr/~lambe_a1/C.html`

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 width=600,int height=782;
  int image[width*height];
  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`.