Exercices sur les répertoires et les fichiers : Génération aléatoire d'image de visage

On souhaite écrire un programme `face` générant aléatoirement une image de visage en composant par superposition des images de cranes, mentons, oreilles, bouches, nez, yeux, sourcils, et optionnellement lunettes, moustaches, barbes et cheveux. On utilisera pour cela la librairie `MagickWand` (avec le compilateur `gcc`, car `clang` semble avoir un problème de compatibilité avec cette librairie). Les ressources sont des fichiers PNG de même dimension 221x347 regroupés par catégories dans des sous-répertoires de noms fixes à l'intérieur d'un répertoire de base.

```
#define MIN_SUBDIR_COUNT  7
#define MAX_SUBDIR_COUNT 11

char * subDirNames []= {
  "cranium", "chin", "ears", "nose", "mouth", "eyes", "eyebrows",
  "beard", "moustache", "hair", "glasses"
};
```

L'utilisateur spécifie le répertoire de base en argument de la ligne de commande, ainsi que la graine initialisant le générateur de nombre pseudo-aléatoire. Le programme affiche le visage généré ou le sauve dans un fichier si un troisième argument est spécifié.

```
$ ./face
Usage: ./face BASEDIR SEED [FILE]
$ ./face ./face-features/221x347 $RANDOM
```

Le programme principal et les modules

```
#include <wand/magick-wand.h>        // version 6
//#include <MagickWand/MagickWand.h> // version 7
...
int main (int argc, char * argv[]) {
  if (argc != 1+2 && argc != 1+3) {
    fprintf (stderr, "Usage: %s DIR SEED [FILE]\n", argv[0]);
    exit (1);
  }
  char * baseDirName= argv[1];
  int seed= atoi (argv[2]);
  char * filename= (argc == 1+3) ? argv[3] : NULL;
  srand (seed);
  MagickWandGenesis();
  MagickWand * face= Image_randomFace (baseDirName, subDirNames,
                                       MIN_SUBDIR_COUNT, MAX_SUBDIR_COUNT);
  if (filename == NULL) MagickDisplayImage(face, getenv ("DISPLAY"));
  else if ( ! MagickWriteImage  (face, filename )) {
    fprintf (stderr, "Error: failed to write image to '%s'\n", filename);
  }
  DestroyMagickWand (face);
  MagickWandTerminus();
  return 0;
}
```

Il s'agit essentiellement de programmer la fonction `Image_randomFace()` qui choisit un élément de visage au hasard par sous-répertoire de `DIR`, sauf pour les quatre sous-répertoires des éléments optionnels où l'on peut aussi ne rien choisir avec une probabilité arbitraire.

Outre le module `Main` du programme principal, il y a trois modules : `Path` pour la manipulation de chemins, `DirContent` pour l'obtention du contenu d'un répertoire, et `Image` pour la génération du visage avec `MagickWand`.

Le module `Path`

La fonction `readdir()`, qui lit le contenu d'un répertoire, ne donne que le nom de chaque fichier, et non son chemin complet incluant le répertoire. Si l'on veut ce chemin, il faut le construire soi-même. Implémenter la fonction `Path_catenate()` qui retourne la concaténation d'un chemin `dirName` et d'un nom `name`, séparés par un slash. Par exemple la concaténation de `"./face-features/221x347"` et `"moustache"` est `"./face-features/221x347/moustache"`. Le résultat est alloué dynamiquement et devra donc être libéré avec `free()` :

```
char * Path_catenate (char const dirName [], char const name []);
```

La fonction `readdir()` qui lit le contenu d'un répertoire retourne `dot= "."` et `dotdot= ".."` parmi ses résultats (répertoire courant et parent). Afin de s'en débarasser, on écrit une fonction pour les détecter facilement. Implémenter le prédicat `Path_isDots()` qui teste si un chemin est `"."` ou `".."` :

```
bool Path_isDots (char const path []);
```

On peut vouloir se débarrasser d'autres fichiers listés par `readdir()`. Pour cela, nos fonctions de listing seront paramétrées avec un prédicat de type `PathPredicate` qui indique si un fichier doit apparaître dans le listing. Si l'on souhaite tout conserver, on peut utiliser le prédicat `Path_isAny()` qui retourne toujours `true`.

```
typedef bool PathPredicate (char const dirName[], char const name[]);
```

```
bool Path_isAny (char const dirName[], char const name[]) {
  (void) dirName; (void) name; return true;
}
```

Implémenter le prédicat `Path_isReadablePng()` qui teste si un fichier est régulier, peut être ouvert en lecture, et possède la signature PNG dans ses 8 premiers octets, c'est-à-dire la séquence de bytes : `(137, 80, 78, 71, 13, 10, 26, 10)`. Utiliser pour cela `stat()`, `fopen()`, et `fread()` :

```
bool Path_isReadablePng (char const dirName[], char const name[]);
```

Le module `DirContent`

Implémenter dans un premier temps la fonction `Dir_printEntries()` qui affiche dans `file` le listing du répertoire préalablement ouvert `dir` de nom `dirName`. Seules les entrées vérifiant le prédicat `filter` sont listées, à raison d'une entrée par ligne. De plus, `dot= "."` et `dotdot= ".."` n'apparaissent pas dans le listing :

```
void Dir_printEntries (DIR * dir, char const dirName[],
                       PathPredicate * filter, FILE * file);
```

La fonction précédente est difficilement utilisable, car l'ordre des entrées obtenues par `readdir()` successifs est arbitraire. On souhaite donc mémoriser le contenu dans une structure `DirContent` vue plus loin, afin de pouvoir trier les entrées. La structure alloue dynamiquement deux tableaux : un pour les entrées, et un pour la totalité des noms de ces entrées. Le premier a donc autant de cases de type `char *` que d'entrées, et le second autant de cases de type `char` que la somme de tailles des noms des entrées, `'\0'` terminaux inclus. Les cases du premier tableau pointent vers le début des chaînes à l'intérieur du second tableau. Afin d'allouer ces deux tableaux, il faut connaître leurs tailles `entryCount` et `charCount` que l'on réunit dans la structure `DirSize` ci-dessous :

```
typedef struct DirSize {
  size_t entryCount;
  size_t charCount; // including all '\0'
} DirSize;
```

Leur calcul implique un premier parcours du répertoire. La fonction `Dir_findSize()` se charge de se travail en retournant une structure de type `DirSize`. Lorsqu'elle se termine, elle repositionne le curseur de lecture où il se trouvait :

```
DirSize Dir_findSize (DIR * dir, char const dirName[], PathPredicate * filter);
```

La fonction `DirContent_read()` ouvre un répertoire de nom `dirName` et mémorise son contenu filtré par le prédicat `filter`. Elle retourne la structure de type `DirContent` ci-dessous. En cas d'échec d'ouverture, le champ `error` est mis à `true`. Sinon, le champ `error` est mis à `false`, le champ `size` est calculé avec `Dir_findSize()`, les deux tableaux `entries` et `chars` sont alloués dynamiquement avec les tailles correspondantes, et ils sont remplis grâce à un second parcours avec `readdir()`.

```
typedef struct DirContent {
  bool error;
  DirSize size;
  char ** entries;
  char * chars;
} DirContent;
```

```
DirContent DirContent_read (char const dirName[], PathPredicate * filter);
```

Les fonctions `DirContent_clean()` et `DirContent_cleanMany()` libèrent respectivement les ressources allouées par une ou plusieurs structures de type `DirContent`. (Rappelons que lorsque le champs `error` vaut `true`, rien n'a été alloué) :

```
void DirContent_clean (DirContent content);
void DirContent_cleanMany (DirContent contents[], size_t contentCount);
```

La fonction `DirContent_readMany()` lit `subDirCount` sous-répertoires de noms `subDirNames[]` situés dans un même répertoire de nom `baseDirName`, et stocke leurs contenus dans les `subDirCount` cases de `contents[]`. Le nombre d'échecs à l'ouverture est retourné par la fonction :

```
size_t DirContent_readMany (char const baseDirName[],
                            char * const subDirNames[], size_t subDirCount,
                            PathPredicate filter, DirContent contents[]);
```

On souhaite trier les entrées des contenus obtenus par `DirContent_read()` et `DirContent_readMany()`. En utilisant `qsort()`, implémenter les fonction de tri `DirContent_sort()` et `DirContent_sortMany()`. Seul le champ `entries` de `Content` est trié. Le champ `chars` n'est pas modifié.

```
void DirContent_sort (DirContent content);
void DirContent_sortMany (DirContent contents[], size_t contentCount);
```

L'utilisation de `qsort()` requiert de coder une fonction de comparaison `DirContent_compareEntry()`. Utiliser d'abord `strcmp()` pour la comparaison des chaînes dans cette fonction et observer comment les entrées sont triées. Opter ensuite pour `strcoll()` et aligner la catégorie `LC_COLLATE` avec sa variable d'environnment via `setlocale()` dans `DirContent_sort()` le temps du tri (restaurer l'ancienne valeur de la catégorie après le tri), et observer l'impact de cette modification.

```
int DirContent_compareEntry (void const * data1, void const * data2);
```

Le module `Image`

Une fois la librairie `MagickWand` initialisée avec `MagickWandGenesis()`, on peut créer une image de fond uni, ainsi que charger une image depuis le disque comme le montrent les fonctions `Image_create()` et `Image_read()` ci-dessous. Pour la couleur `bgColor`, on peut utiliser le blanc `"#FFFFFF"` :

```
MagickWand * Image_create (int width, int height, char const bgColor[]) {
  MagickWand * image= NewMagickWand();
  PixelWand * bgPixel= NewPixelWand();
  PixelSetColor (bgPixel, bgColor);
  MagickNewImage (image, width, height, bgPixel);
  DestroyPixelWand (bgPixel);
  return image;
}
```

```
MagickWand * Image_read (char const path[]) {
  MagickWand * image= NewMagickWand();
  MagickReadImage (image, path);
  return image;
}
```

La fonction `MagickCompositeImage (target, source, OverCompositeOp, 0,0)` ou
la fonction `MagickCompositeImage (target, source, OverCompositeOp, false, 0,0)`,
selon qu'on utilise la version 6 ou 7 de la libraire, permet de superposer une image `source` sur une image `target`, et la fonction `DestroyMagickWand (image)` permet de détruire une image.

Avec ces fonctions, implémenter `Image_randomFace()` qui retourne une image générée aléatoirement à partir des éléments de visage des sous-répertoires du répertoire de base `baseDirName`.

```
MagickWand * Image_randomFace (char const baseDirName [],
                               char * const subDirNames [],
                               size_t minSubDirCount, size_t maxSubDirCount);
```