Langage C : IO et mémoire

Jean-François Lalande et Pierre Wilke

1 Écriture et lecture binaire

On considère le code ci-dessous, qui écrit sous forme binaire une struture `struct mystruct` :

```
struct mystruct {
  int x;
  char * name;
};

int main(){
  FILE * f;
  f = fopen("mydata.blob", "wb");

  struct mystruct * m = malloc(sizeof(struct mystruct));
  m->x = 4;
  m->name = malloc(sizeof(char)*100);
  strcpy(m->name, "coucou");
  printf("mystruct->x = %d\n", m->x);
  printf("mystruct->name = %s\n", m->name);

  fwrite(m, sizeof(struct mystruct), 1, f);
  fclose(f);
}
```

Exercice 1 Visualisez le contenu du fichier `mydata.blob` avec un éditeur hexadécimal (`xxd`, `hexdump`, ...). Vous constaterez le *padding* des champs de la structure permettant d'accélerer les accès à ces champs. On voit que la chaîne de caractère `"coucou"` est absente. Pourquoi ?

Exercice 2 Dans le même programme (à la suite du code précédent), réalisez le code de lecture qui lit binairement, à l'aide de `fread`, les données du fichier `mydata.blob` et qui réinstancie une structure de type `struct mystruct` pointée par `m2` (attention à bien allouer de la mémoire pour `m2` : `m` et `m2` pointent sur deux zones de mémoire différentes). Affichez les données de la structure pointée par `m2`.

Exercice 3 Après l'écriture du fichier et avant la lecture du fichier, modifiez la chaine de caractère de l'attribut `name` du pointeur `m` à l'aide de `strcpy`. Que constatez-vous sur les champs de la structure `m2` ?

Exercice 4 Pour corriger ce problème, sans changer la structure, nous proposons d'écrire, en plus, dans le fichier, la chaine pointée par `m->name`, puis de la relire. Corrigez le code dans ce sens.

2 Lecture/Écriture binaire d'un ABR

Un Arbre Binaire de Recherche est un arbre dont chaque nœud possède deux fils et une clef. Sur un tel arbre, on maintient la propriété suivante : pour un nœud donné n, toutes les clefs des fils à gauche de ce nœud n sont inférieures à la clef de n ; à droite les clefs sont supérieures. Ce type de structure permet de représenter des ensembles d'éléments, et de pouvoir en extraire une liste triée facilement.

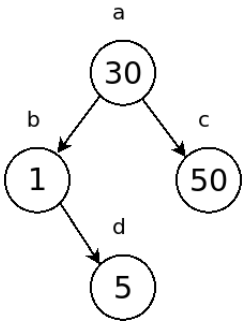
Dans cette partie, on propose de réaliser le stockage d'un arbre binaire dans un fichier, au format binaire. La difficulté réside dans la relecture et la réinstantiation des structures en mémoire.

Pour débuter, on partira de l'arbre donné dans le listing 1.

Listing 1 – Arbre à 4 nœuds

```
typedef struct arbre {
  int x;
  struct arbre * fg;
  struct arbre * fd;
} A;

int main(){
  A d = { .x = 5, .fg = NULL, .fd = NULL };
  A c = { .x = 50, .fg = NULL, .fd = NULL }
  A b = { .x = 1, .fg = NULL, .fd = &d };
  A a = { .x = 30, .fg = &b, .fd = &c };
}
```



2.1 Écriture / Lecture

Exercice 5 Réalisez une fonction récursive `displayArbre` qui affiche l'arbre. Pour chaque nœud on affichera sa valeur et l'adresse de ce nœud.

On utilisera le spécifieur de format `%p` pour afficher la valeur d'un pointeur.

La sortie attendue est la suivante, pour l'arbre donné ci-dessus :

```
Noeud 30 (0x7ffe7c03f510)
  Noeud 1 (0x7ffe7c03f4f0)
    Noeud 5 (0x7ffe7c03f4b0)
  Noeud 50 (0x7ffe7c03f4d0)
```

Exercice 6 La fonction précédente est pratique pour observer la structure de l'arbre, mais pas tellement pour vérifier que l'arbre est correctement trié. Écrivez une fonction `displaySorted` qui affiche les clés d'un arbre dans l'ordre, en tirant partie de la structure d'un ABR et de la propriété que ces arbres possèdent.

Sur l'arbre précédent, on doit obtenir :

```
1 5 30 50
```

Exercice 7 Réalisez une fonction récursive `writeArbre` qui écrit binairement les nœuds de l'arbre dans un fichier `arbre.blob`. L'algorithme écrira la racine, puis le sous arbre gauche récursivement, puis le sous arbre droit récursivement.

Exercice 8 Après avoir sauvegardé votre arbre, modifiez les valeurs de x de chaque nœud de l'arbre en mémoire à -1, par exemple en faisant :

```
a.x = −1; b.x = −1;
c.x = −1; d.x = −1;
```

Exercice 9 Affichez l'arbre pour vérifier que tous les nœuds sont à -1.

Exercice 10 Prenez cinq minutes (un papier et un crayon…) pour réfléchir à la fonction qui va gérer la réinstanciation des nœuds de l'arbre du fichier. Qu'est ce qui va être difficile ? (Pensez à ce que vous avez appris dans la première partie de ce BE !)

Exercice 11 Réalisez une fonction récursive `readArbre` qui reconstruit l'arbre depuis le fichier `arbre.blob` en allouant l'espace nécessaire. Affichez votre arbre afin de tester votre implémentation.

Exercice 12 A-t-on utilisé les valeurs des pointeurs `fg` et `fd` stockées dans le fichier ? Que se passerait-il si on lisait ce fichier sur un ordinateur Big Endian ?

2.2 Instanciation d'arbre

Exercice 13 Réalisez une fonction `A * insert(A * a, int x)` qui permet d'insérer correctement la valeur `x` dans l'arbre `a`.

Note 1 : Votre fonction doit fonctionner même si l'arbre passé en paramètre est `NULL`.

Note 2 : Un ABR représente des ensembles de valeur, donc si la valeur que vous devez insérer est déjà présente, ne l'insérez pas à nouveau.

Exercice 14 Réalisez la fonction `A * loadFromFile(char * myfile)` qui renvoie l'arbre instancié à partir du fichier `myfile` qui contiendra un entier par ligne :

```
30
1
...
```

Exercice 15 Testez la sauvegarde et la restauration binaire de vos arbres générés à partir de fichiers texte que l'on pourra générer en faisant (dans un terminal sous Linux (ça marche peut-être aussi sous MacOS, si vous avez fait un jour `brew install coreutils`)) :

```
shuf -i 0-100 -n 50 > test.txt
```

Si vous êtes sous Windows, soit vous êtes un as de PowerShell, soit vous utiliserez le fichier `test.txt` que nous vous fournirons.