Piscine - C - Tek1

Sujet Jour 14

Responsable Astek astek_resp@epitech.eu

Table des matières

Consignes

`Cette suite d'exercices concerne uniquement les tek2ed et tek3s`
`Amis tek1, inscrivez vous au module B1 - Igraph, regardez le cours d'introduction. Amusez vous bien !`

- Le sujet peut changer jusqu'à une heure avant le rendu.
- Vos exercices doivent être à la norme.
- Vous ne devez avoir de `main()` dans aucun fichier de votre répertoire de rendu.
- Pour chaque repertoire de chaque exercice nous allons compiler vos fichiers avec la commande `cc -c *.c`, ce qui va générer tous les fichiers `.o` que nous allons ensuite linker un par un en y ajoutant notre `main.c` :

```
$> cd ex\_01
$> cc -I/afs/epitech.net/users/group/login/rendu/include/ -c *.c
$> cc *.o ~moulinette/main_ex_01.o -o ex01 -L/afs/epitech.net/users/group/login/rendu/lib/ -lmy
$> ./ex01
[...]
```

- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
  Si un seul de vos fichiers empêche la compilation avec `*.c`, la moulinette ne pourra pas vous corriger et vous aurez `0`. Vous avez donc tout intéret à effacer vos rendus d'exercices ne fonctionnant pas.
- Pensez à en discuter sur le forum piscine !
- Travaillez en local !
  C'est-à-dire que pour chaque exercice vous devez le compiler sur votre compte linux puis, une fois qu'il fonctionne, le copier sur votre compte AFS.
  Ceci dans le simple but de ne pas surcharger les serveurs car vous êtes nombreux.

  Indices `Faites-vous un script shell pour copier vos fichiers sur l'AFS`

- Rendu :
  `/afs/epitech.net/users/group/login/rendu/piscine/Jour_14`
- Votre bibliothèque sera utilisée durant la phase de 'linkage'.

Détails techniques

- Pour les exos d'aujourd'hui, on utilisera la structure suivante :

```
1   typedef struct s_btree
2   {
3       struct s_btree *left;
4       struct s_btree *right;
5       void *item;
6   } t_btree;
```

Exercice 01 : btree_create_node

- Écrire la fonction `btree_create_node` qui alloue un nouvel élément, initialise son `item` à la valeur du paramètre et tous les autres éléments à 0.
- L'adresse de la node créée est renvoyée.
- Elle devra être prototypée de la façon suivante :

```
1   t_btree *btree_create_node(void *item);
```

- Rendu :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_00/`

Exercice 02 : btree_apply_prefix

- Écrire la fonction `btree_apply_prefix` qui applique la fonction passée en paramètre à l'`item` de chaque node, en parcourant l'arbre de manière `préfixe`.
- Elle devra être prototypée de la façon suivante :

```
1    void btree_apply_prefix(t_btree *root, int (*applyf)(void *));
```

- Rendu :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_01/`

Exercice 02 : btree_apply_infix

- Écrire la fonction `btree_apply_infix` qui applique la fonction passée en paramètre à l'`item` de chaque node, en parcourant l'arbre de manière `infix`.
- Elle devra être prototypée de la façon suivante :

```
1   void btree_apply_infix(t_btree *root, int (*applyf)(void *));
```

- Rendu :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_02/`

Exercice 03 : btree_apply_suffix

- Écrire la fonction `btree_apply_suffix` qui applique la fonction passée en paramètre à l'`item` de chaque node, en parcourant l'arbre de manière `suffix`.
- Elle devra être prototypée de la façon suivante :

```
1    void btree_apply_suffix(t_btree *root, int (*applyf)(void *));
```

- Rendu :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_03/`

Exercice 04 : btree_insert_data

- Écrire la fonction `btree_insert_data` qui insert l'élément `item` dans un arbre. L'arbre passé en paramètre sera trié, c'est à dire que pour chaque `node` tous les élements inférieurs se situent dans la partie gauche, et tous les éléments supérieurs ou égaux à droite. On enverra en paramètre une fonction de comparaison ayant le même comportement que `strcmp`.
- Elle devra être prototypée de la façon suivante :

```
1   void btree_insert_data(t_btree **root, void *item, int (*cmpf)(void *, void *));
```

- Rendu :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_04/`

Exercice 05 : btree_search_item

- Écrire la fonction `btree_search_item` qui retourne le premier élèment correspondant à la donnée de référence passée en paramètre. L'arbre devra être parcouru de manière `infix`. Si l'élément n'est pas trouvé, la fonction devra retourner `NULL`.
- Elle devra être prototypée de la façon suivante :

```
1   void *btree_search_item(t_btree *root, void *data_ref, int (*cmpf)(void *, void *));
```

- Rendu :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_05/`

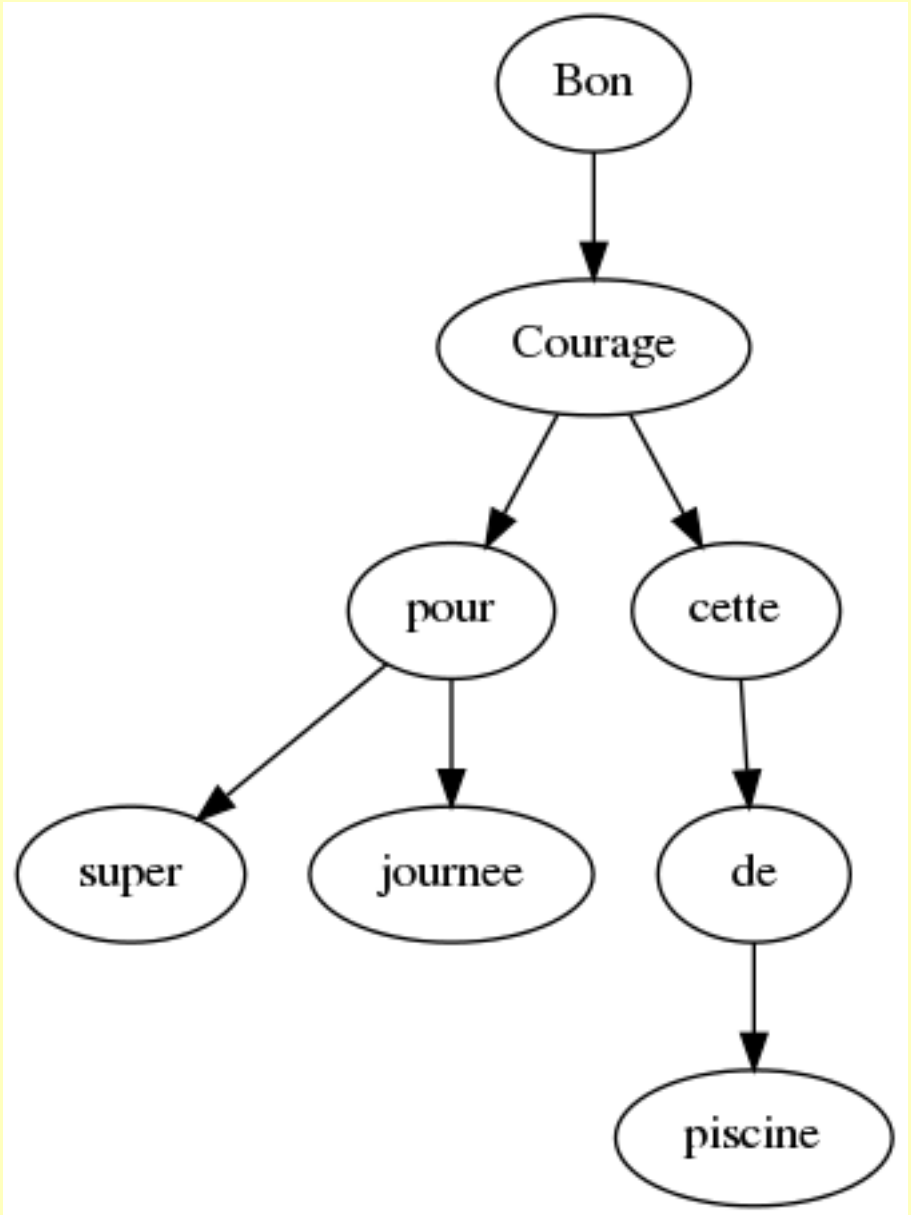
Exercice 06 : btree_level_count

- Écrire la fonction `btree_level_count` qui retourne la taille de la plus grande branche passée en paramètre.
- Elle devra être prototypée de la façon suivante :

```
1   int btree_level_count(t_btree *root);
```

- Rendu :
`/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_06/`

Indices



`Dans cet exemple, on retournera 5.`

Exercice 07 : btree_apply_by_level

- Écrire la fonction `btree_apply_by_level` qui applique la fonction passée en paramètre à chaque noeud de l'arbre. L'arbre doit être parcouru étage par étage. La fonction appelée prendra trois paramètres :
  - Le premier paramètre, de type `void *`, correspond à l'item du node
  - Le second paramètre, de type `int`, correspond au niveau sur lequel on se trouve : 0 pour le root, 1 pour ses enfants, 2 pour ses petits-enfants...
  - Le troisième paramètre, de type `int` vaut 1 s'il s'agit du premier `node` du niveau, sinon 0
- Elle devra être prototypée de la façon suivante :

```
1  void btree_apply_by_level(t_btree *root, void (*applyf)(void *item, int current_level, int
       is_first_elem))
```

- Rendu :
  `/afs/epitech.net/users/group/login/rendu/piscine/Jour_14/ex_07/`

Indices

Pour un arbre représenté par l'image ci-dessus, la fonction applyf serait appelée successivement avec les paramètres suivants :

- "Bon", 0, 1
- "Courage", 1, 1
- "pour", 2, 1
- "cette", 2, 0
- "super", 3, 1
- "journee", 3, 0
- "de", 3, 0
- "piscine", 4, 1

Détails techniques

- Nous allons maintenant travailler avec des arbres rouges et noirs.

```
1  typedef struct rb_node
2  {
3      enum RB_COLOR { RB_BLACK, RB_RED } color;
4      void *data;
5      struct rb_node *left;
6      struct rb_node *right;
7  } t_rb_node;
```

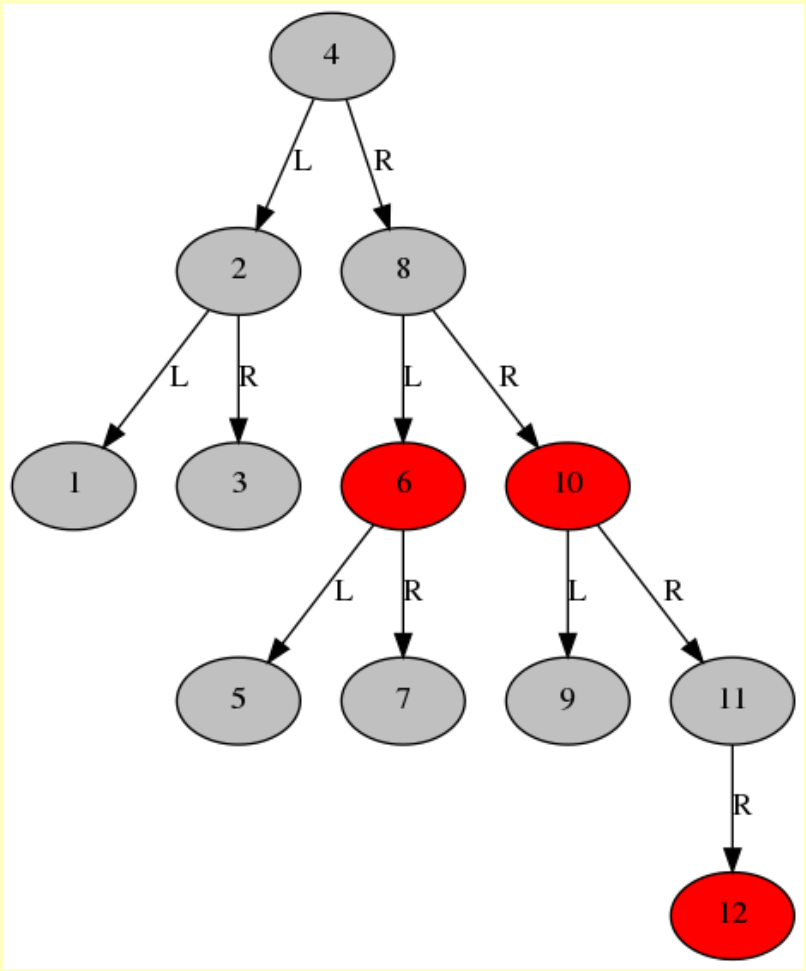
exercice 08 : rb_insert

- écrire la fonction `rb_insert` qui ajoute une nouvelle donnée dans l'arbre de manière à ce qu'il continue de respecter les contraintes d'un arbre rouge et noir. le paramètre `root` pointe sur le noeud racine de l'arbre. lors du premier appel, il pointe sur `null`. on enverra aussi en paramètre une fonction de comparaison ayant le même comportement que `strcmp`.
- elle devra être prototypée de la façon suivante :

```
1   void rb_insert(struct rb_node **root, void *data, int (*cmpf)(void *, void *));
```

- rendu :
  `/afs/epitech.net/users/group/login/rendu/piscine/jour_14/ex_08/`

Indices



exemple d'un arbre rouge et noir

exercice 09 : rb_remove

- écrire la fonction `rb_remove` qui supprime une donnée dans l'arbre de manière à ce qu'il continue de rsepecter les contraintes d'un arbre rouge et noir. le paramètre `root` pointe sur le noeud racine de l'arbre. On enverra aussi en paramètre une fonction de comparaison ayant le même comportement que `strcmp`, ainsi qu'une pointeur sur fonction `freef` qui sera appelée avec en paramètre l'élément de l'arbre à supprimer.
- elle devra être prototypée de la façon suivante :

```
1   void rb_remove(struct rb_node **root, void *data, int (*cmpf)(void *, void *), void (*freef)(
        void *));
```

- rendu :
  `/afs/epitech.net/users/group/login/rendu/piscine/jour_14/ex_09/`