Algorithmique et Structures de données 1

L2 2021-2022

Travaux Pratiques 5

Site du cours : `https://defelice.up8.site/algo-struct.html`
Les exercices marqués de (@) sont à faire dans un second temps.
Un fichier écrit en langage C se termine conventionnellement par `.c`.
Une commande de compilation est `gcc fichier_source1.c fichier_source2.c fichier_source3.c`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall -Wextra` pour demander au compilateur d'afficher plus de Warnings
- `-std=c11` pour compiler selon la norme C11
- `-g -fsanitize=address` pour compiler avec information de débogage et en interdisant la plupart des accès a une zone mémoire non reservée.

Exemple : `gcc -Wall fichier1.c -o monprogramme`

Ce TP porte sur des arbres binaires étiquetés. On utilisera la structure suivante. **Respecter les noms des types et des champs.**

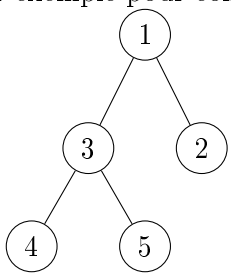
```
typedef struct s_noeud_t
{
   int v; // étiquette du noeud (v pour valeur)
   struct s_noeud_t* g; // pointeur vers la racine du sous-arbre  gauche
   struct s_noeud_t* d; // pointeur vers la racine du sous-arbre droit
} noeud_t;

// l'arbre vide aura la valeur NULL
```

Exercice 1. *Opérations de bases*

Définir les fonctions suivantes :

- `noeud_t* consA(noeud_t* gau,noeud_t* droit,int etiquette)`. Qui assemble deux sous-arbres pour produire un nouvel arbre. Renvoie l'adresse de la nouvelle racine. Attention à la gestion de la mémoire.
- `void liberer(noeud_t* a)` qui libère la mémoire occupée par les noeuds de l'arbre `a`.

Par exemple pour construire l'arbre :



On peut utiliser l'appel suivant :

`consA(consA(consA(NULL,NULL,4),consA(NULL,NULL,5),3),consA(NULL,NULL,2),1)` (Les arbres vides sont implicites et ne sont pas représentés sur la figure)

Exercice 2. *Parcours !*

Écrire les fonctions suivantes :

1. `void parcourirPref(noeud_t* a)` qui imprime le résultat d'un parcours préfixe de l'arbre.
2. `void parcourirPost(noeud_t* a)` qui imprime le résultat d'un parcours postfixé de l'arbre.

3. `void parcourirInfix(noeud_t* a)` qui affiche dans l'ordre infixe, les étiquettes de l'arbre avec des parenthèses.

Exercice 3. *Compter*

Écrire les fonctions suivantes.

1. `int taille(noeud_t* arbre)` qui renvoie le nombre de nœuds (étiqueté) d'un arbre.
2. `int nbFeuilles(noeud_t* arbre)` qui renvoie le nombre de feuilles d'un arbre. (Une feuille est un noeud qui n'a que des fils vides).
3. `int hauteur(noeud_t* a)` qui renvoie la hauteur de l'arbre `a`. On convient qu'un arbre vide possède une hauteur de `-1` et que la hauteur est la plus grande distance des feuilles à la racine.

Exercice 4. *Trouver*

Construire une fonction `int estDans(noeud_t* a,int v)` qui renvoie 1 si la valeur `v` est une étiquette de l'arbre `a`, 0 sinon.

Exercice 5. *Construction par parcours préfixe*

1. Construire une fonction `void construirePref(noeud_t** a)` permettant de construire un arbre à partir d'entiers positifs lus au clavier. Les nœuds seront décrits par un parcours en en profondeur préfixe, un fils vide sera codé par -1. Ainsi la suite
   `1 3 4 -1 -1 5 -1 -1 2 -1 -1`
   décrit l'arbre du début du sujet.
   Aide : On pourra utiliser `scanf` pour lire les nombres entrés.
2. `int construirePrefT(neud_t** A,int* tab)` qui construit l'arbre à partir d'un tableau de nombres `tab`. La fonction renvoie le nombre de cases lues pour construire l'arbre. On utilise le même code que la question précédente.