

## SDD TP3 : ARBRE

Nicolas Cavani & Adrien Leduque  
ZZ1 ISIMA  
enseignant : Yves Jean Daniel

mai 2020

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Arborescence du projet</b>	<b>2</b>
<b>3</b>	<b>Makefile</b>	<b>3</b>
<b>4</b>	<b>Structure de données</b>	<b>4</b>
<b>5</b>	<b>Mode d'emploi</b>	<b>5</b>
<b>6</b>	<b>Fonction et Procédure</b>	<b>6</b>
<b>7</b>	<b>Traces</b>	<b>8</b>
<b>8</b>	<b>Cas d'exécution</b>	<b>17</b>

## 1 Introduction

L'objectif de ce TP est de coder une bibliothèque de fonction permettant la manipulation de la structure en arbre. Cette bibliothèque permet de :

- Créer un arbre à partir d'une représentation algébrique stockée dans un fichier
- Écrire la représentation postfixée (et préfixée) de l'arbre
- Rechercher un élément dans l'arbre
- Ajouter un fils à élément dans l'arbre

Les fils d'un point sont triés par ordre alphabétique.

## 2 Arborescence du projet

Le projet contient plusieurs fichiers sources.

- pilefile : contient les fonctions de manipulation des piles et des files que l'on a implémenté dans le TP2
- arbre : contient les fonctions de manipulation des arbres
- affichageArbre : contient les fonctions d'affichages des arbres
- represPost : contient les fonctions de création de la représentation postfixée des arbres
- fctChaines : contient des fonctions de manipulation des chaînes de caractères

### 3 Makefile

Un Makefile est un fichier constitué de plusieurs règles de la forme :

cible : dépendances

commandes

#### 3.1 Cibles :

all : regroupe dans ses dépendances l'executable à produire.

clean : permet de supprimer tous les fichiers intermédiaires.

.PHONY : permet de reconstruire les dépendances de la cible (dépendante de .PHONY), dans le cas où des fichiers porteraient le même nom.

#### 3.2 Variables :

CFLAGS : regroupe les options de compilation.

-Wall -Wextra : warnings de compilation

-g : Génère les informations de débogage.

-MMD : permet la génération des fichiers .d pour les dépendances.

SRC : contient la liste des fichiers sources du projet. La commande wildcard permet l'utilisation du joker \* dans la définition d'une variable.

OBJ : contient la liste des fichiers objets. OBJ est rempli à partir de SRC.

\$(patsubst pattern, replacement, text) : Trouve les mots dans text qui correspondent au pattern et les remplace par replacement.

DEP : fichiers .d contenant les dépendances associées à un fichier .o du même nom.

#### 3.3 Variables internes :

\$@ : Le nom de la cible.

\$< : Le nom de la première dépendance.

\$^ : La liste des dépendances.

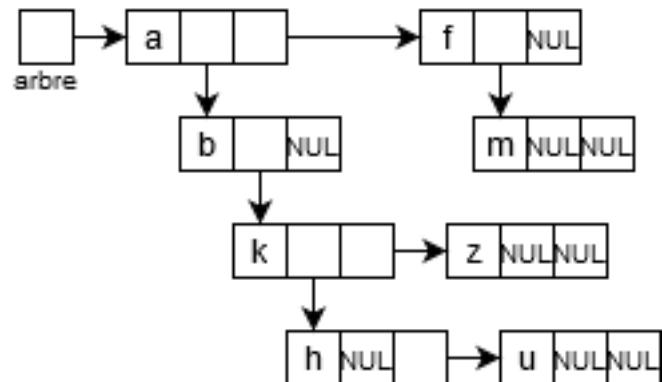
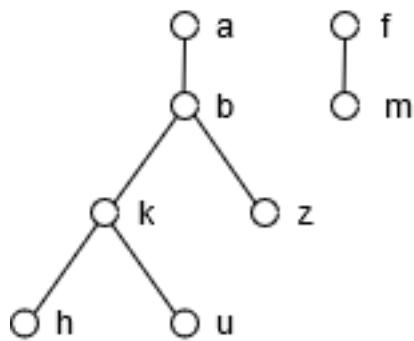
@ : Permet de ne pas afficher la commande dans la console.

## 4 Structure de données

Un arbre est formé de plusieurs noeuds représentés par la structure nommée "elemArbre\_t". Chaque noeud de l'arbre contient une valeur, un pointeur sur un frère et un pointeur sur un fils. La structure contient donc :

- La valeur du noeud
- Un pointeur sur le premier fils
- Un pointeur sur le frère suivant

elemArbre		
valeur	fils	frere
a	NUL	NUL



## 5 Mode d'emploi

Le programme fonctionne de la manière suivante :

Exécuter le programme : ./bin/executable

Une fois l'exécutable lancé, le programme effectue les instructions contenu dans le main.

Peuvent être utilisées les fonctions suivantes :

- **Fonctions de création de la notation algébrique**
  - *recupNotaAlgebrique*
- **Fonctions de création et de traitement de l'arbre**
  - *creerElemArbre*
  - *creerArbreNotaAlgebrique*
  - *rechercherValeur*
  - *insererFils*
  - *libererArbre*
- **Fonctions d'affichage de l'arbre**
  - *afficherArbre*
  - *actuTabFreres*
  - *afficherValeur*
  - *afficherArbrePost*
  - *afficherArbrePref*
- **Fonctions liées à la représentation postfixée**
  - *creerRepresPost*
  - *ajouterValeurRepres*
  - *compterFils*
  - *afficherRepres*
- **Fonctions utilitaires de traitement de chaînes**
  - *charFinChaine*
  - *tailleChaine*
  - *copierChaine*
  - *agrandirChaine*
  - *ecoTailleChaine*
  - *entierEnChaine*
  - *inverserChaine*

### Fonctions Importantes :

**recupNotaAlgébrique** : Lis le fichier passé en paramètre, récupère la notation algébrique qui y est contenue et la stock dans une liste contiguë.

**creerArbreNotaAlgébrique** : Crée un arbre à partir d'une notation algébrique passée en paramètre.

**rechercheValeur** : Cherche dans l'arbre passé en paramètre, une valeur.

**insererFils** : Insère dans l'arbre une nouvelle valeur fils à un élément de l'arbre si il existe.

**libérerArbre** : Libère l'arbre de la mémoire.

**afficherArbrePost** : Affiche les éléments de l'arbre dans l'ordre postfixé.

**creerRepresPost** : Crée la représentation postfixée de l'arbre et la stock dans une liste contiguë.

## 6 Fonction et Procédure

### 6.1 Fonctions de création et de traitement de l'arbre

#### 6.1.1 recuperNotaAlgebrique

Récupère la notation algébrique stockée dans un fichier et la met dans une chaîne de caractères.

En entrée : filename (char \*) : nom du fichier contenant la notation

En sortie : notation (char \*) : chaîne de caractères, notation algébrique

#### 6.1.2 creerElemArbre

Créer un nouvel élément de type elemArbre\_t

En entrée : void

En sortie : elemArbre (elemArbre\_t \*) : élément créé

#### 6.1.3 creerArbreNotaAlgebrique

Créer un arbre à partir d'une notation algébrique

En entrée : notation (char \*) : notation algébrique

En sortie : arbre (elemArbre\_t \*) pointeur sur la racine de l'arbre

#### 6.1.4 rechercherValeur

Recherche une valeur dans l'arbre

En entrée :

- arbre (elemArbre\_t \*) : pointeur sur la racine de l'arbre

- valeur (char) : valeur à rechercher dans l'arbre

En sortie : cour (elemArbre\_t \*) : pointeur sur l'élément contenant la valeur, NULL si la valeur n'a pas été trouvée

#### 6.1.5 insererFils

Insère un fils à un élément

En entrée :

- arbre (elemArbre\_t \*) : pointeur sur la racine de l'arbre

- valeurPere (char) : valeur à chercher dans l'arbre pour y insérer le fils

- nouvValeur (char) : valeur du nouvel élément fils

En sortie : codeErreur (char) : booléen d'erreur, 1 si erreur 0 sinon

#### 6.1.6 libererArbre

Libere la memoire, free tous les éléments de l'arbre

En entrée : arbre (elemArbre\_t \*\*) : pointeur sur la racine de l'arbre passé par adresse

En sortie : codeErreur (char) : booléen d'erreur, 1 si erreur 0 sinon

## 6.2 Fonctions liées à la représentation postfixée

### 6.2.1 creerRepresPost

Créer la représentation postfixée de l'arbre et la stock dans un tableau contiguë

En entrée : arbre (elemArbre\_t \*) : pointeur sur la racine de l'arbre

En sortie : represPost (char \*) : pointeur sur le tableau contiguë contenant la représentation postfixée

## 6.3 ajouterValeurRepres

Remplie le tableau (la representation) avec la valeur courante

En entrée :

- repres (char \*\*) : chaîne de caractere, représentation (prefixée ou postfixée)

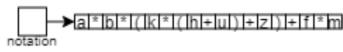
- elemArbre (elemArbre\_t \*) : élément courant dont la valeur est à insérer dans la représentation
- nbElem (int \*) : compteur, indice pour remplir le tableau (par adresse)
- taille (int \*) : taille du tableau repres (par adresse)

## 7 Traces

### 7.1 creerArbreNotaAlgebrique

#### En entrée

chaine de caractères contenant la représentation algébrique de l'arbre que l'on souhaite créer

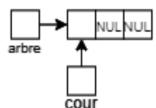


#### Initialisation

On créer le pointeur sur l'arbre que l'on retourne à la fin de la fonction

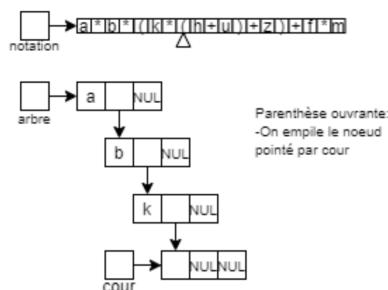
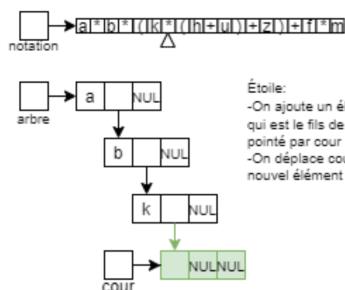
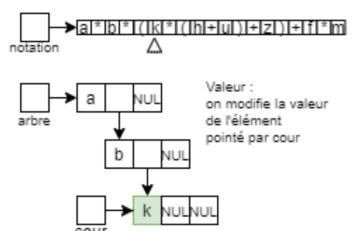
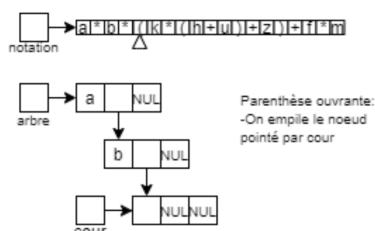
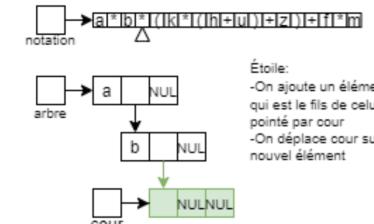
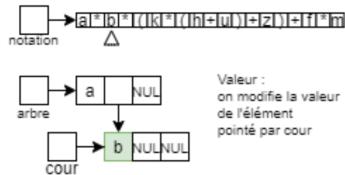
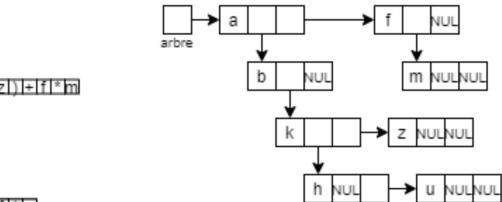
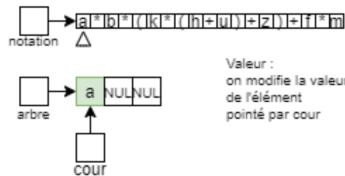


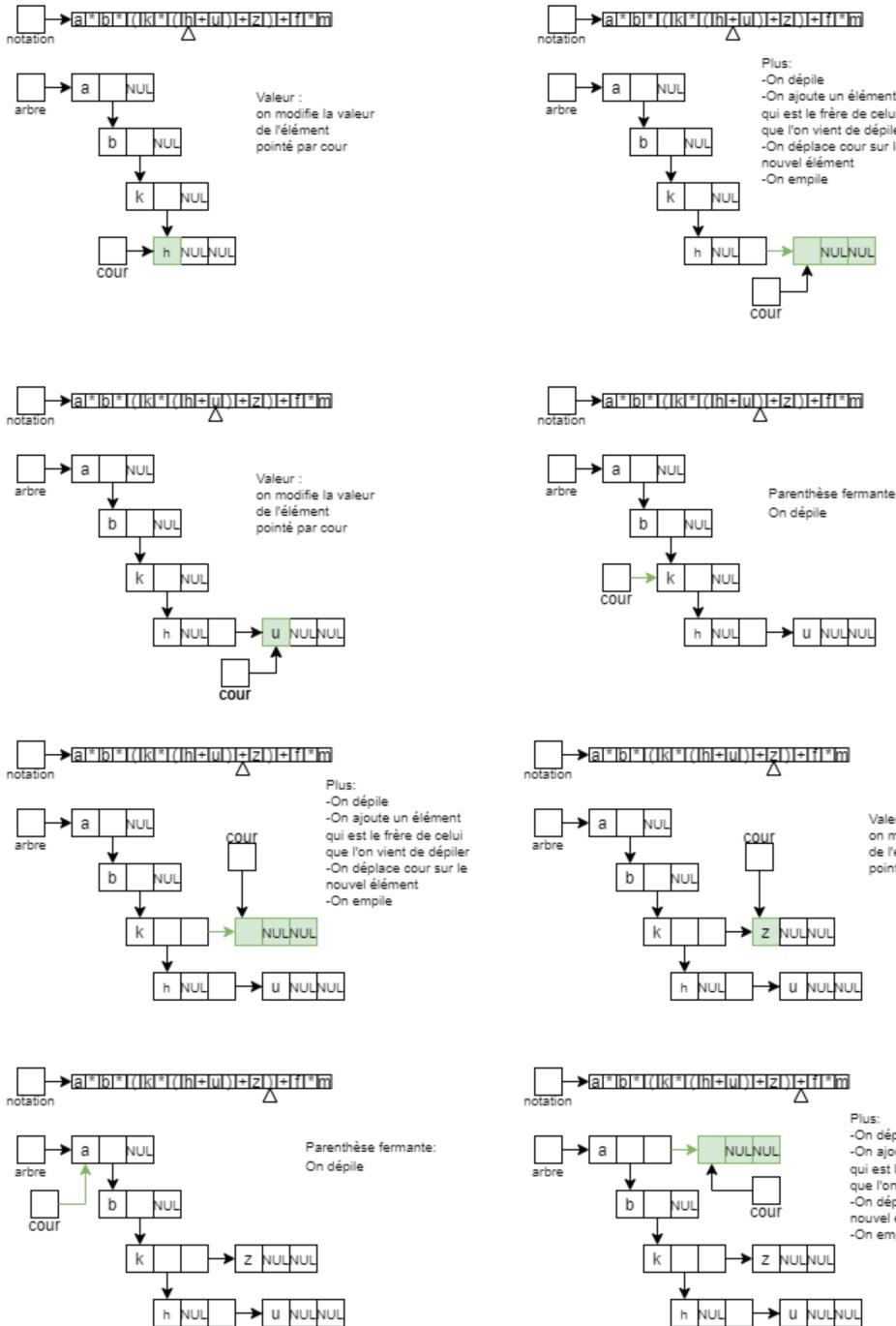
On créer le premier élément de l'arbre

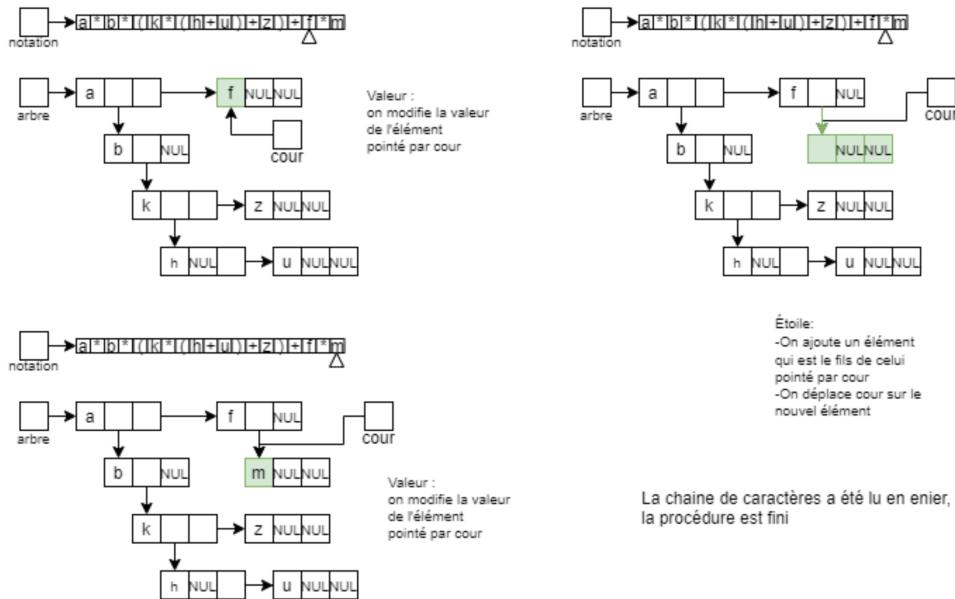


On créer le pointeur qui parcourra l'arbre pour le créer

On empile le premier élément





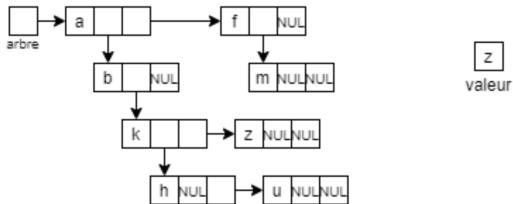


## 7.2 rechercherValeur

### En entrée

-l'arbre dans lequel on souhaite réaliser la recherche

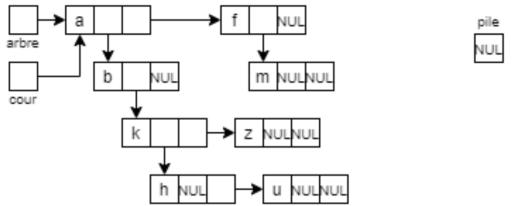
-La valeur que l'on cherche



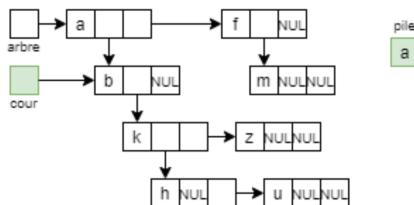
### Initialisation

On créer le pointeur qui parcourra l'arbre pour la recherche

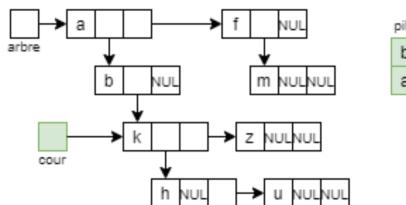
On créer la pile



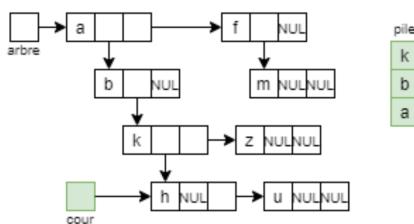
- On empile le noeud
- On part sur le fils



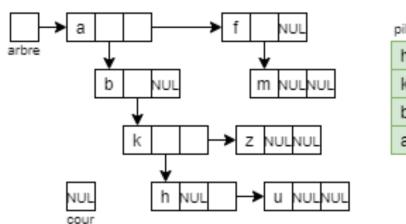
- On empile le noeud
- On part sur le fils



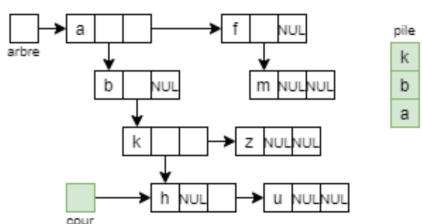
- On empile le noeud
- On part sur le fils



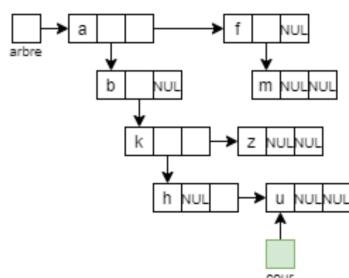
- On empile le noeud
- On part sur le fils



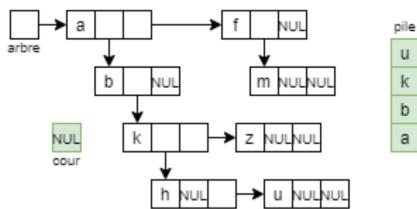
- On dépile



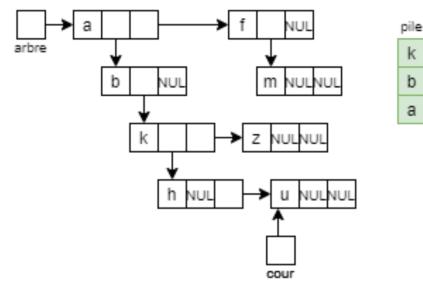
- On part sur le frere



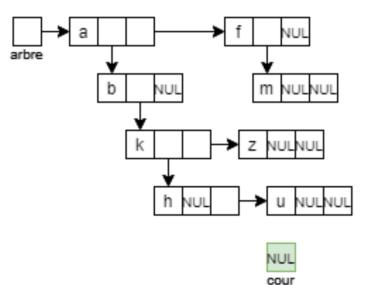
-On empile le noeud  
-On part sur le fils



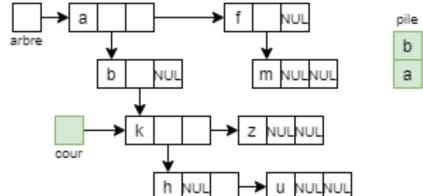
On dépile



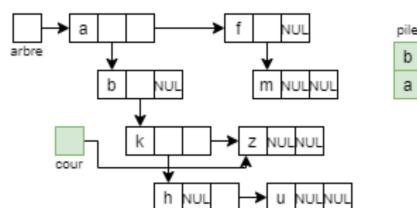
On part sur le frere



On dépile



On part sur le frere



On a trouvé le noeud qui contient la valeur recherchée  
On libère la pile  
La procédure est terminée

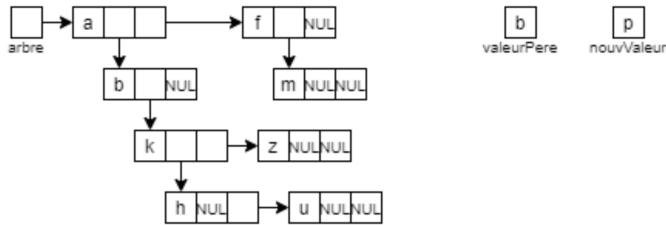
### 7.3 insererFils

#### En entrée

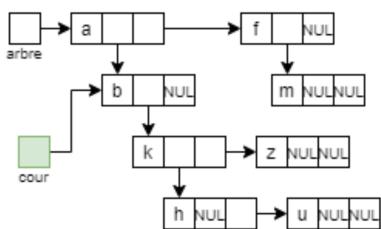
-l'arbre dans lequel on souhaite réaliser l'ajout du fils

-La valeur du père

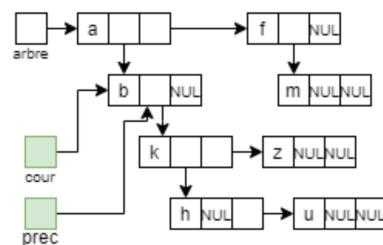
-La valeur du fils



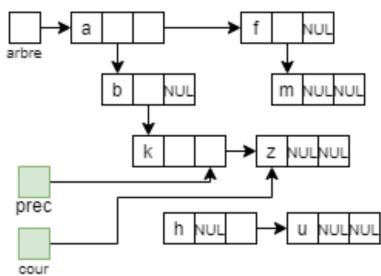
On recherche le père  
Appel de la procédure rechercherValeur



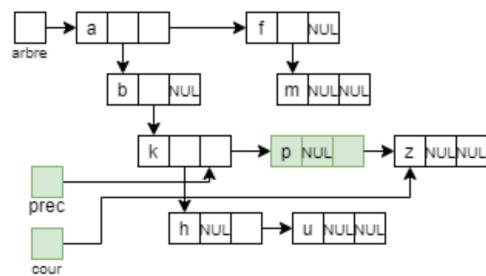
On crée un pointeur sur l'emplacement où l'on veut ajouter le nouvel élément



On se place sur le bon frère (par ordre alphabétique)



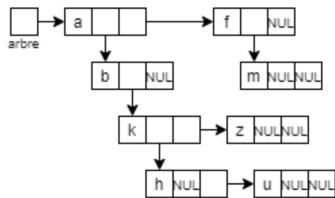
On insère le nouveau élément



## 7.4 libérerArbre

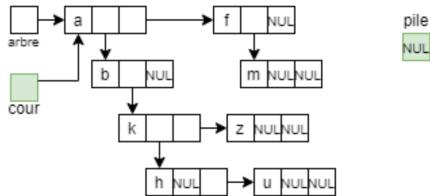
### En entrée

L'arbre que l'on souhaite libérer

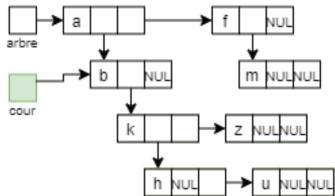


### Initialisation

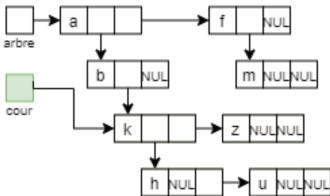
On crée un pointeur sur le premier noeud de l'arbre  
On créer la pile



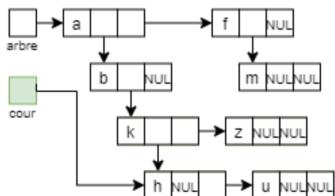
-On empile  
-On part sur le fils



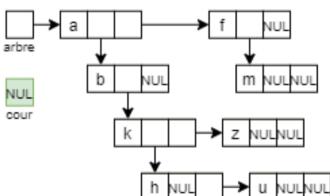
-On empile  
-On part sur le fils



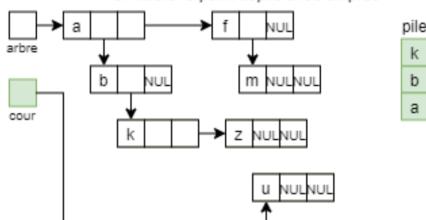
-On empile  
-On part sur le fils



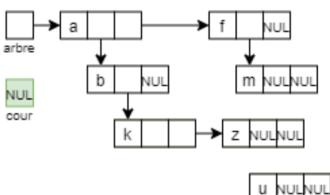
-On empile  
-On part sur le fils

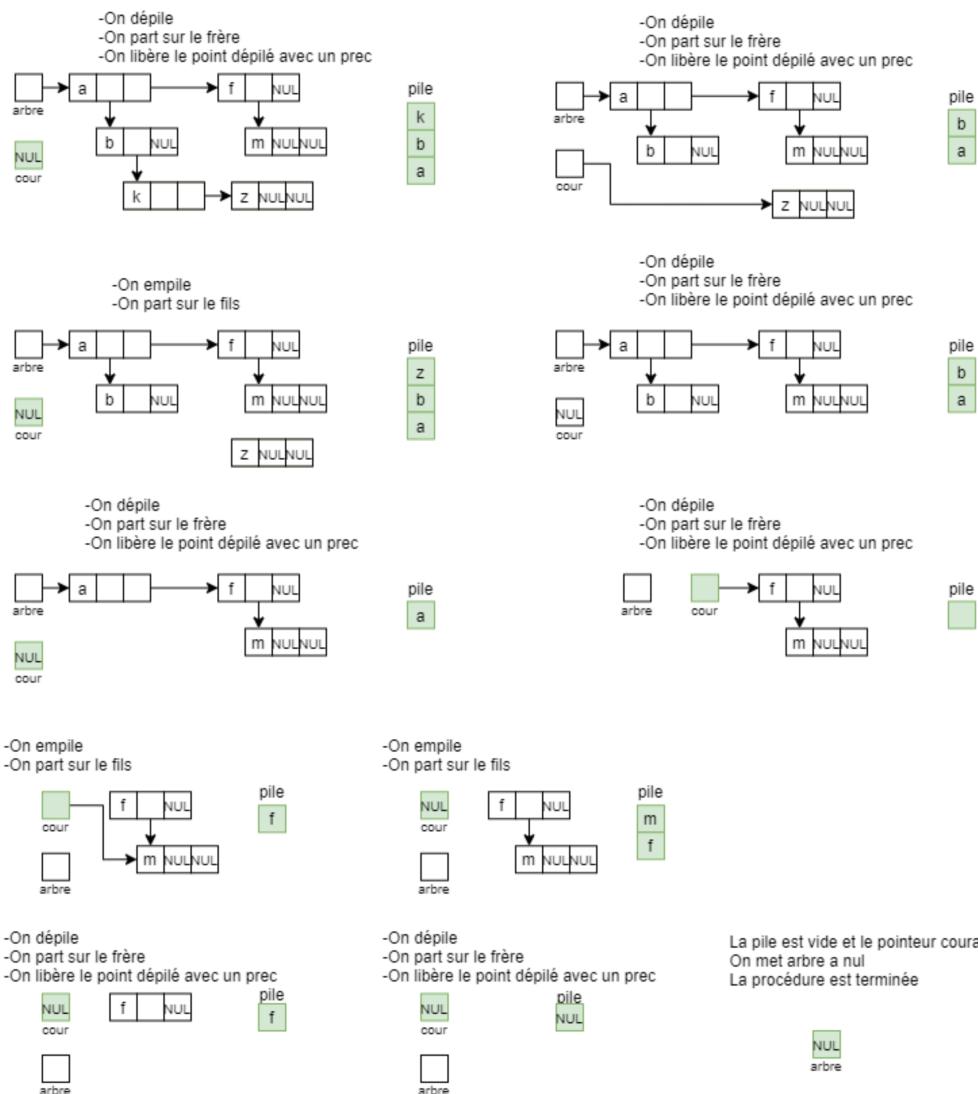


-On dépile  
-On part sur le frère  
-On libère le point déplié avec un prec



-On empile  
-On part sur le fils





## 7.5 afficherArbrePost

Même parcourt de l'arbre que pour la libération, mais en remplaçant la libération de l'élément par son affichage

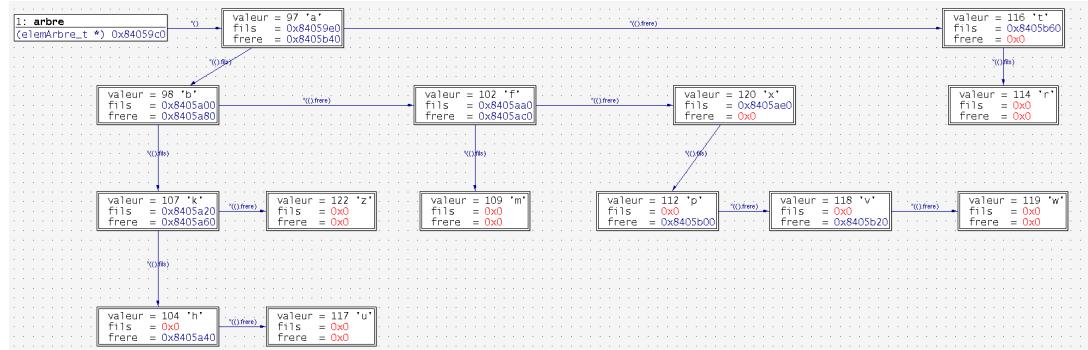
## 8 Cas d'exécution

### 8.1 Cas Général

$$a^*(b^*(k^*(h+u)+z)+f^*m+x^*(p+v+w))+t^*r$$

Ce cas général contient tous les cas suivant :

- Le dernier fils n'a pas de frère
- Le dernier fils a un frère
- Le dernier fils a plus de 1 frère
- Le dernier frère n'a pas de fils
- Le dernier frère a un fils
- Le dernier frère a plus de 1 fils
- un noeud au milieu de l'arbre a un frère
- un noeud au milieu de l'arbre n'a pas de fils
- un noeud au milieu de l'arbre n'a pas de frère
- un noeud au milieu de l'arbre a un fils
- un noeud au milieu de l'arbre a plus de 1 fils
- un noeud au milieu de l'arbre a plus de 1 frère



Représentation postfixée :

(h,0)(u,0)(k,2)(z,0)(b,2)(m,0)(f,1)(p,0)(v,0)(w,0)(x,3)(a,3)(r,0)(t,1)

### 8.2 arbre vide

Un arbre vide :



### 8.3 arbre a un seul élément

Un arbre a un seul élément : a

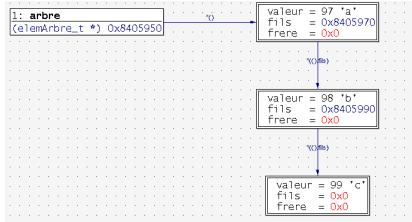


Représentation postfixée :

(a,0)

## 8.4 arbre que des fils

Un arbre contenant que des fils : a\*b\*c



Représentation postfixée :  
 $(c,0)(b,1)(a,1)$

## 8.5 arbre que des frère

Un arbre contenant que des frère : a+b+c



Représentation postfixée :  
 $(a,0)(b,0)(c,0)$