Travaux Pratiques d'AP3 n°2

—Licence Informatique, 3ième année, 1ier semestre—

Les structures arborescentes

Le but de ce TP est de d'initier à la programmation et à la manipulation des structures arborescentes.

▸ **Exercice 1. (Calculs élémentaires sur les arbres binaires)**

1. *Définissez trois exemples d'arbre avec 7 noeuds.*
2. *Écrivez une fonction* `size` *qui calcule la taille d'un arbre binaire.*
3. *Écrivez une fonction* `height` *qui calcule la hauteur d'un arbre binaire.*
4. *Écrivez une fonction* `size_at_height` *qui, pour une hauteur donnée, calcule le nombre de noeuds dans un arbre binaire.*
5. *Écrivez une fonction qui calcule le nombre de noeuds internes d'un arbre binaire.*
6. *Écrivez une fonction qui calcule le nombre de feuille d'un arbre binaire.*

▸ **Exercice 2. (Parcours sur les arbres binaires)**

1. *Le bord gauche d'un arbre est le chemin obtenu en ne suivant que des liens gauche à partir de la racine. Écrire une fonction* `left_bodrer` *qui donne le bord gauche d'un arbre binaire sous la forme d'une liste des noeuds parcourus.*
2. *Pour chaque parcours en profondeur (préfixe, infixe, postfixe), écrivez des fonctions* `to_list_prefix`, `to_list_infix`, `to_list_postfix` *qui rangent les noeuds dans une liste selon leur ordre d'apparition. Pour écrire ces fonctions, utilisez la concaténation de deux listes.*

▸ **Exercice 3.** ***(Parcours en largeur d'un arbre)*** *L'objectif de cet exercice est d'implanter une fonction qui réalise un parcours en largeur d'un arbre binaire. Le parcours en largeur d'un arbre binaire est un parcours de ses noeuds niveau par niveau, en suivant, par exemple, l'ordre hiérarchique.*

1. *À partir d'exemples, donnez quelques parcours en largeur d'arbres binaires.*
2. *À la question précédente, vous avez dû réaliser quelle est la difficulté de ce parcours. En utilisant une file d'arbres binaires (Un module implantant les files vous est fourni sur UPdago), donnez un algorithme de parcours en largeur.*

3. *Écrivez une fonction* `largeur : 'a tree -> 'a list` *qui renvoie la liste des noeuds d'un arbre parcouru en largeur et qui implante votre algorithme en utilisant les modules des files et des arbres des questions précédentes.*

▸ **Exercice 4. (Manipulation d'arbres de syntaxe abstraite)** *Les expressions mathématiques comme* `2 * x + sin( -x )` *se représentent facilement sous forme d'arbre (pas nécessairement binaire). Ces arbres s'appellent des arbres de syntaxe abstraite et sont à la base des manipulations des textes des programmes par les compilateurs.*

Dans le cadre de cet exercice, nous allons nous restreindre à des arbres binaires et en conséquence les expressions manipulées ne seront construites qu'avec les opérateurs binaires $+$, $*$, $-$ *et* $/$. *Les expressions contiennent des constantes qui sont des entiers et des variables dont le nom est représenté avec une chaîne de caractères.*

1. *Donnez trois exemples d'expressions représentées sous forme d'arbre binaire.*
2. *Pour représenter une expression sous la forme d'un arbre binaire, on a besoin de définir un type qui permet de représenter les valeurs aux noeuds. Définissez un type somme* `nodelbl` *qui permet représenter les valeurs dont vous avez besoin et donnez les termes qui représentent les expressions de la question 1.*
3. *Afin de pouvoir afficher les arbres de syntaxe abstraite, écrivez une fonction* `string_of_nodelbl` *qui transforme une valeur du type* `nodelbl` *en une chaîne de caractères. Comme nous l'avons vu en cours, l'utilisation d'une valeur d'un type somme est traitée par un filtrage.*

   Utilisez la fonction `show` *du module* `Btree` *pour afficher les arbres correspondant à vos exemples.*
4. *Tous les arbres que vous pouvez construire, ne représentent pas des expressions, pourquoi ? Donnez des exemples d'arbres qui ne représentent pas une expression. Écrivez une fonction* `well_formed` *qui vérifie si un arbre représente bien une expression.*

   Pour les questions qui suivent, on suppose que les arbres binaires donnés en arguments aux fonctions sont bien formés.
5. *Afin de pouvoir afficher une expression sous la forme habituelle écrivez une fonction* `string_of_expr` *qui transforme une expression en une chaîne de caractères. Quel l'ordre de parcours d'arbre que vous utilisez ?*
6. *On veut maintenant calculer l'expression qui est la dérivée d'une expression par rapport à une variable. Écrivez une fonction* `derive` *qui prend en arguments un arbre binaire représentant une expression et un nom de variable et qui construit l'expression qui est la dérivée par rapport à cette variable. Voici quelques rappels qui vont seront utiles*
   $(u + v)' = u' + v'$
   $(u * v)' = u' * v + u * v'$
   $(u/v)' = (u' * v - u * v')/v^2$
7. *On veut maintenant évaluer une expression. Pour cela il faut définir une valeur pour chaque variable qui apparaît dans une expression. Cela se fait avec une liste de couples où le premier composant est le nom d'une variable et le second composant est la valeur qui lui correspond. On appelle une telle liste, une liste d'associations. Écrivez une*

fonction `valassoc` *qui en argument un nom de variable et une liste d'assocaitions et qui renvoie la valeur associée à cette variable. Utilisez la fonction* `valassoc` *pour écrire une fonction* `eval` *qui évalue une expression donnée sous la forme d'un arbre binaire.*

8. *(Facultatif) Dans la question 4, on a vu qu'il existe des arbres binaires construit avec le type* `nodelbl` *qui ne représentent pas des expressions. Proposez un type d'arbres qui permet d'éviter cet inconvénient.*