Programmation et Structures de Données 1995-96
TD 5
Arbres et Forêts

Frédéric Vivien, bureau 306, e-mail: Frederic.Vivien

20 octobre 1995

1 Arbres binaires

Un noeud d'un arbre binaire est soit vide, soit contient un entier (la clé) et deux arbres binaires fils.

- Définir une structure de données "arbres binaires".
- Écrire une fonction *make_tree* qui prend comme argument deux entiers m et n et qui retourne un arbre binaire qui contient inclusivement tous les entiers entre m et n.
- Écrire une fonction *print_prefix_tree* qui prend en argument un arbre et qui écrit l'arbre de manière préfixe. Exemple: l'arbre (17, $fils\_gauche$, $fils\_droit$) sera représenté:

```
+
  fils_gauche
  fils_droit
```

- Écrire une fonction *print_infix_tree* qui prend en argument un arbre et qui écrit l'arbre de manière infixe. Exemple: l'arbre (17, $fils\_gauche$, $fils\_droit$) sera représenté:

```
  fils_gauche
+
  fils_droit
```

- Écrire une fonction *print_postfix_tree* qui prend en argument un arbre et qui écrit l'arbre de manière postfixe. Exemple: l'arbre (17, $fils\_gauche$, $fils\_droit$) sera représenté:

```
  fils_gauche
  fils_droit
+
```

2 Parcours d'arbres

On suppose que l'on dispose d'une structure de données appelée *bag* dans laquelle on peut stocker des arbres. Sur cette structure de données, nous disposons de trois fonctions: *is_empty_bag()* qui renvoie un booléen codant l'état du sac; *add_bag(arbre, sac)* qui ajoute l'arbre *arbre* au sac *sac* et qui renvoie le nouveau sac; *get_bag(sac)* qui enlève un arbre du sac *sac* si celui-ci n'est pas vide et qui renvoie l'arbre enlevé ainsi que le nouveau sac.

Écrire une fonction *traversal* qui prend un sac *sac* comme argument et qui parcoure ce sac. Cette fonction ne fait rien si *sac* est vide. Sinon, elle enlève un arbre *arbre* de *sac*. Si cet arbre est vide, le parcours se poursuit sur le reste du sac. Sinon, la clé de la racine de l'arbre est affichée, et la traversée se poursuit sur le reste du sac *sac* auquel a été rajouté les deux arbres fils de la racine de *arbre*!

3 "Sacs d'arbres"

Définir le type *bag* comme une liste d'arbres.

Écrire les fonctions *get_bag(sac)* et *add_bag(arbre, sac)*. Pour cette dernière fonction, utiliser une fonction *coin()* qui renvoie un booléen et insérer le nouvel élément en fonction du résultat de l'invocation de la fonction *coin()*: si *coin()* renvoie *true* insérer le nouvel élément en tête, sinon l'insérer après le premier élément de *sac* grâce à un appel récursif.

Qu'obtient-on si *coin()* renvoie toujours *true*? Qu'obtient-on si *coin()* renvoie toujours *false*?

On pourra utiliser aussi une fonction *coin()* aléatoire grâce à la fonction *random_int limite* du module *random*.

4 Question subsidiaire

La fonction *traversal* termine-t-elle toujours si dans le cas où *sac* et *arbre* sont non vides, deux exemplaires de chacun des fils de la racine de l'arbre *arbre* sont rajoutés au reste du sac *sac* après l'affichage de la racine de *arbre*? Pourquoi?