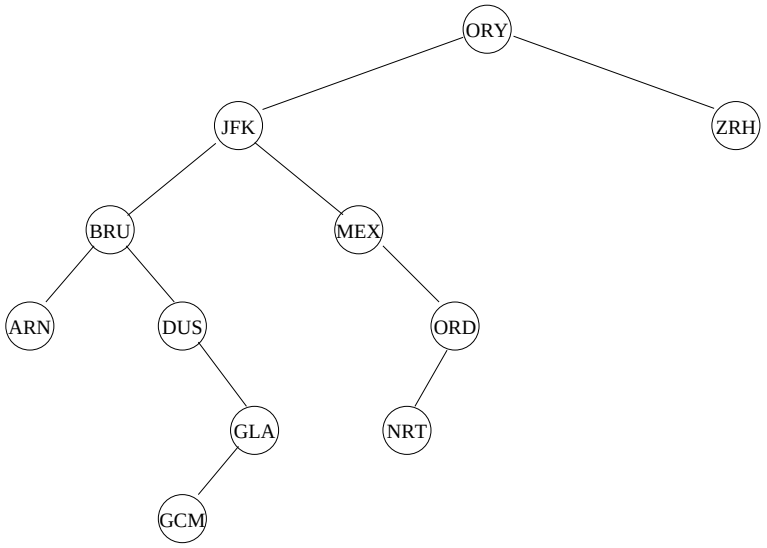
Master mention informatique
Spécialité STL

MI030 – Analyse des programmes et sémantique (APS)
Second examen réparti
Lundi 4 juin 2012, 13h45 – 15h45

Directives

1. Le contrôle dure 2h00.
2. Le contrôle comporte X questions sur Y pages en tout.
3. Les points attribués à chaque question sont donnés à titre indicatif et totalisent 20 points sur l'ensemble du contrôle.
4. Tous les documents sont autorisés.
5. Tous les appareils électroniques sont **prohibés** (y compris les téléphones portables, les assistants numériques personnels et les agendas électroniques).

Question 1. Un arbre de recherche binaire est une structure de données arborescente permettant de lier des clés à des valeurs. Chaque nœud porte une clé et une valeur. L'arbre de recherche binaire garantit ensuite que pour tout nœud n, toutes les clés dans son sous-arbre de gauche sont strictement plus petites et toutes les clés dans son sous-arbre de droite sont plus grandes ou égales à la clé de n. La figure suivante montre un arbre de recherche binaire contenant uniquement des clés qui sont ici les codes en trois lettres des aéroports internationaux : (6 points)



Nous avons vu en TD comment il est possible de définir une structure de données comme une λ-expression capturant les valeurs des champs sous la forme de paramètres, puis retournant une λ-expression capable de prendre une commande et de l'exécuter sur la structure de données. Nous vous demandons de définir la structure de données arbre de recherche binaire sous cette forme. Votre arbre de recherche binaire devra pouvoir exécuter les commandes suivantes :

- `isEmpty` : retourne vrai si l'arbre est vide et faux sinon.
- `find` : prend une clé et retourne la valeur qui lui correspond dans l'arbre ou la clé elle-même si elle n'existe pas dans l'arbre.
- `insert` : prend une clé et une valeur et les insère au bon endroit dans l'arbre.

Programmation contractuelle en BOPL

En programmation par objets, la programmation contractuelle consiste à introduire des assertions sur les classes et les méthodes qui seront vérifiées à l'exécution. Chaque classe possède une assertion, son invariant portant sur les variables d'instance de la classe, qui doit être vérifiée en tout temps. Chaque méthode possède deux assertions :

- une pré-condition, portant sur les paramètres et les variables d'instance, qui doit être vérifiée à l'entrée de la méthode, et
- une post-condition, portant sur la valeur de retour et les variables d'instance, qui doit être vérifiée à la sortie de la méthode.

Lorsqu'une assertion est fausse, une erreur est déclenchée. Voici un exemple de l'ajout à faire à BOPL en syntaxe concrète :

```
program
  class Reservoir is
  vars
    Int capacite, contenu ;
  invariant
    (self.contenu < self.capacite or self.contenu = self.capacite)
    and (self.contenu > 0 or self.contenu = 0)
  methods
    Void remplir(Int r)
    pre r < (self.capacite - self.contenu) or r = (self.capacite - self.contenu)
    post true
    begin
      self.contenu := self.contenu + r ;
    end
    Int leContenu()
    pre true
    post return = self.contenu
    begin
      return self.contenu ;
    end
  end
begin
  ...
end
```

Concernant l'invariant, il doit être valide en tout temps, mais en réalité il faudra qu'il soit valide à l'entrée et à la sortie des méthodes. Au sens de BOPL, les assertions sont des expressions à résultat booléen.

Question 2. Proposez les ajouts à la syntaxe abstraite de BOPL pour intégrer les invariants, pré-conditions et post-conditions. Donnez les modifications à la sémantique de BOPL pour prendre en compte ces assertions. Notez que votre solution n'a pas à tenir compte de l'héritage. (14 points)

Fin du contrôle.