Algorithmique 1 : Devoir Surveillé 3

Arbres

Durée : 60mn
Sans documents

1 Arbres Binaires

Exercice 3.1 *Parcours hiérarchique*
Écrire une fonction `parcours` qui affiche les sommets d'un arbre binaire par niveaux. Sur l'arbre de la Fig. 1, on affichera `/ 9 * + - 1 2 4 1`.

Exercice 3.2 *Arbre syntaxique d'expression*
L'arbre syntaxique d'une expression est défini par la règle récursive suivante :
"placer l'opérateur à la racine, l'arbre pour l'expression correspondant au premier opérande à gauche et l'arbre pour l'expression correspondant au deuxième opérande à droite".
Une illustration est donnée sur la Fig. 1.
Tout comme les écritures infixées et postfixées des expressions mathématiques, l'arbre syntaxique est une manière de représenter une expression mathématique.
Écrire une fonction `construire` qui construit l'arbre syntaxique d'une expression à partir d'une repésentation postfixée donnée en entrée. Ainsi la fonction `construire` à partir de l'écriture postfixée de `E`, E_{pf} `= 9 1 2 + 4 1 - * /`, produira l'arbre syntaxique de la Fig. 1.
Pour simplifier on supposera que les opérateurs sont dyadiques $+, -, *$ et $/$.

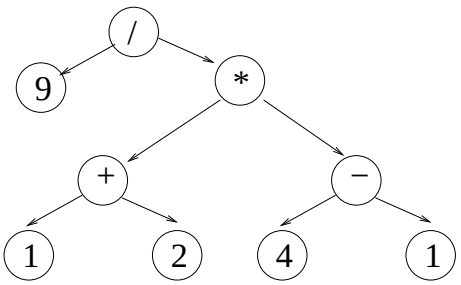

FIG. 1 – Arbre syntaxique de l'expression $E = 9/((1+2)*(4-1))$

2 Arbres Planaires

Exercice 3.3 *Primitives*
Pour l'implémentation du type `sommetArbrePlanaire` donnée en annexe D, écrire la primitive
`fonction ajouterFils(ref S:sommetArbrePlanaire, val x:objet):vide;`

Exercice 3.4 *Parcours*
Écrire une fonction itérative de parcours préfixe d'un arbre de type `sommetArbrePlanaire`.

3 Arbres Binaires de Recherche

Exercice 3.5
Dessiner des arbres binaires de recherche de hauteur `2,3,4,5`, et `6` pour le même ensemble de clés `{1,4,5,10,16,17,21}`.

Exercice 3.6
Démontrer que si un sommet du type `ABR` a deux fils alors son successeur n'a pas de fils gauche et son predecesseur n'a pas de fils droit.

ANNEXE A **Type abstrait** *arbreBinaire*

```
arbreBinaire= curseur;
sommet= curseur;
fonction creerArbreBinaire(val Racine:objet):sommet;
fonction getValeur(val S:sommet):objet;
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
fonction setValeur(ref S:sommet, val x:objet):vide;
fonction ajouterFilsGauche(ref S:sommet, val x:objet):vide;
fonction ajouterFilsDroit(ref S:sommet, x:objet):vide;
fonction supprimerFilsGauche(ref S:sommet):vide;
fonction supprimerFilsDroit(ref S:sommet):vide;
fonction detruireSommet(ref S:sommet):vide;
```

ANNEXE B **Implémentation du type abstrait** *arbreBinaire*

```
cellule=structure
         info:objet;
         gauche: sommet;
         droit: sommet;
         pere: sommet;
       finstructure
sommet= ^cellule;
arbreBinaire= sommet;
```

ANNEXE C **Type abstrait** *sommetArbrePlanaire*

```
sommetArbrePlanaire= curseur;
fonction creerArborescence(val Racine:objet):sommetArbrePlanaire;
fonction valeur(val S:sommetArbreBinaire):objet;
fonction premierFils(val S:sommetArbreBinaire):sommetArbreBinaire;
fonction frere(val S:sommetArbreBinaire):sommetArbreBinaire;
fonction pere(val S:sommetArbreBinaire):sommetArbreBinaire;
fonction setValeur(ref S:sommetArbrePlanaire, val x:objet):vide;
fonction ajouterFils(ref S:sommetArbrePlanaire, val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;
```

Annexe D **Implémentation du type abstrait** *sommetArbrePlanaire*

```
cellule= structure
          info: objet;
          premierFils: sommet
          frere: sommet;
          pere: sommet
         finstructure
sommet= ^cellule;
sommetArbrePlanaire= sommet;
```

Annexe E : **Arbres Binaires de Recherche** *ABR*

On utilise les primitives des arbres binaires.

```
fonction ajouter(ref x: sommet, val e: objet): vide;
fonction supprimer(ref x: sommet): booleen;
```