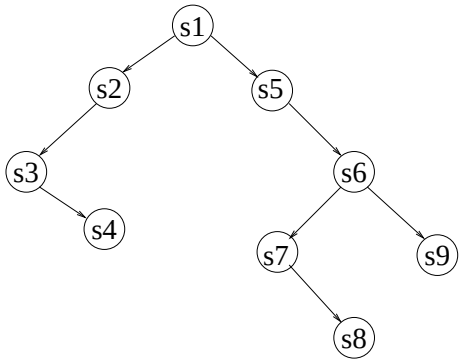
Université Bordeaux 1 LST Mention Informatique Semestre 3 2008-2009

Algorithmique 1 : Devoir Surveillé 3

Arbres

Durée : 40mn
Sans documents.
Les primitives nécessaires à l'écriture des fonctions demandées sont données en annexe.

Exercice 3.1 *Rang d'un sommet dans le parcours par niveaux d'un arbre binaire*
Etant donné un sommet `x` dans un arbre binaire `T` , écrire une fonction `rang` qui calcule la position de `x` dans l'ordre linéaire déterminé par un parcours par niveaux de `T`.
Par exemple pour l'arbre illustré sur la Fig. 1, le parcours par niveaux produit la suite de sommets `s1,s2,s5,s3,s6,s4,s7,s9,s8`. L'appel de la fonction `rang` sur `T` et le sommet `s3` doit renvoyer `4`.
On supposera que le type `arbreBinaire` est implémenté par l'allocation dynamique.



T

FIG. 1 – Arbre binaire

Exercice 3.2 *Arbre planaire*
Dessiner l'arbre planaire construit par la fonction `main`.

```
fonction main(): vide
var s1, s2, s3, s4, s7, s8, s10: sommetArbrePlanaire;
debut
  s1= creerArborescence(1);
  ajouterFils(s1, 2);
  s2= premierFils(s1);
  ajouterFils(s2, 6);
  ajouterFrere(s2,3);
  s3= frere(s2);
  ajouterFrere(s3,4);
  s4= frere(s3);
  ajouterFrere(s4,5);
```

```
  ajouterFils(s4,7);
  s7= premierFils(s4);
  ajouterFrere(s7,8);
  s8= frere(s7);
  ajouterFrere(s8,9);
  ajouterFils(s8,10);
  s10= premierFils(s8);
  ajouterFrere(s10,11);
fin
```

Donner l'algorithme de parcours prefixe d'un arbre planaire et le résultat de son application sur l'arbre planaire `s1`

Exercice 3.3 *Arbre binaire de recherche*
Soit l'arbre binaire de recherche donné sur la Fig. 2.
-Dessiner l'arbre après la suppression du noeud `13`.
-L'opération suppression est-elle "commutative" au sens où la suppression de `x` puis de `y` dans un arbre binaire de recherche produit le même arbre que la suppression de `y` puis de `x`.
Si oui dire pourquoi, sinon donner un contre exemple.

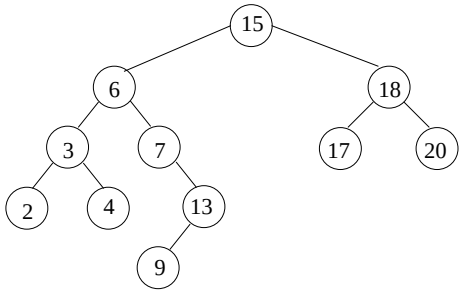


FIG. 2 – ABR

Annexe

```
fonction valeurPile(ref P: pile d'objetPile): objetPile;
fonction pileVide(ref P: pile d'objetPile): bool;
fonction creerPile(ref P: pile d'objetPile): vide;
fonction empiler(ref P: pile d'objetPile, val x: ): vide;
fonction depiler(ref P: pile d'objetPile): vide;
fonction detruirePile(ref P: pile d'objetPile): vide;

fonction valeurFile(ref F: file d'objetFile): objetFile:
fonction fileVide(ref F: file d'objetFile): bool
fonction creerFile(ref F: file d'objetFile): vide;
fonction enfiler(ref F: file d'objetFile, var x: objetFile): vide;
fonction defiler(ref F: file d'objetFile): vide;
fonction detruireFile(ref F: file d'objetFile): vide;

fonction new(ref c: cellule): vide;
fonction delete(ref c: cellule): vide;
fonction valeur(val L: listeDC_car): objet;
fonction debutListe(ref L: listeDC_car): vide;
fonction finListe(ref L: listeDC_car): vide;
fonction suivant(ref L: listeDC_car): vide;
fonction precedent(ref L: listeDC_car): vide;
fonction listeVide(val L: listeDC_car): booleen;
fonction getCleListe(val L: listeDC_car): curseur;
fonction creerListe(ref L: listeDC_car): vide;
fonction insererApres(ref L: listeDC_car, val x: objet): vide;
fonction insererEnTete(ref L: listeDC_car, val x: objet): vide;
fonction supprimerApres(ref L: listeDC_car): vide;
fonction supprimerEnTete(ref L: listeDC_car): vide;
fonction setCleListe(ref L: listeDC_car, val c: curseur): vide;
fonction detruireListe(ref L: listeDC_car): vide;

struct cellule
{
  objet info;
  sommet gauche;
  sommet droit;
  sommet pere;
};
sommet= ^cellule;
arbreBinaire= sommet;
fonction valeur(val S: sommet): objet;
fonction filsGauche(val S: sommet): sommet;
fonction filsDroit(val S: sommet): sommet;
fonction pere(val S: sommet): sommet;
fonction creerArbreBinaire(objet x): sommet;
fonction void ajouterFilsGauche(ref S: sommet, objet x): vide;
```

```
fonction void ajouterFilsDroit(ref S: sommet, objet x): vide;
fonction void supprimerFilsGauche(ref S: sommet): vide;
fonction void supprimerFilsDroit(ref S: sommet): vide;
fonction void supprimerSommet(ref S: sommet): vide;

struct celluleArbrePlanaire
{
  objet info;
  sommetArbrePlanaire premierFils;
  sommetArbrePlanaire frere;
  sommetArbrePlanaire pere;
};
sommetArbrePlanaire= ^celluleArbrePlanaire;
arbrePlanaire= sommetArbrePlanaire;
fonction valeur(val s: sommetArbrePlanaire): objet;
fonction premierFils(val s: sommetArbrePlanaire): sommetArbrePlanaire;
fonction frere(val s: sommetArbrePlanaire): sommetArbrePlanaire;
fonction pere(val s: sommetArbrePlanaire): sommetArbrePlanaire;
fonction creerArborescence(objet racine): arbrePlanaire;
fonction ajouterFils(ref s: sommetArbrePlanaire, objet x): vide;
fonction ajouterFrere(ref s: sommetArbrePlanaire, objet x): vide;
fonction supprimerSommet(ref s: sommetArbrePlanaire): vide;
```