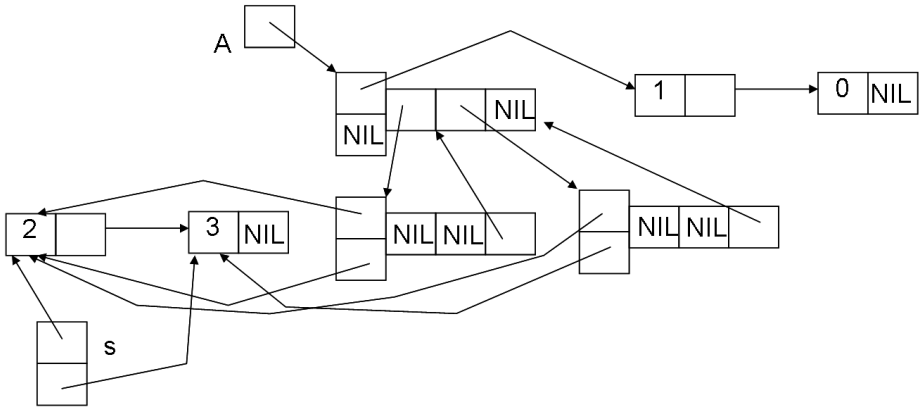
Contrôle continu 3
Pointeurs - Récursivité – Listes – Piles – Files– Arbres
(3 pages)

Questions de cours (5 points)

- Peut-on créer un arbre binaire dont la valeur du sommet est une pile d'entiers?
  oui, le champs info peut contenir tout type défini.
- Quelle structure de données est la plus efficace pour parcourir un arbre en ordre hiérarchique ?
  Une file de sommet (voir le paragraphe parcours hiérarchique)
- Quel parcours existe pour les arbres binaires et n'existe pas pour les arbres planaires ?
  Le parcours infixe
- Dessinez la mémoire, après la suite d'opérations suivante :

```
var A : arbreBinaire de listeSC d'entier;
var s :listeSC d'entier;
creerListe(s) ;
insérerEnTete(s,0);
insérerEnTete(s,1);
A=creerArbreBinaire(s) ;
creerListe(s) ;
insérerEntete(s,2) ;
débutListe(s) ;
ajouterFilsGauche(A,s)
insérerApres(s,3)
suivant(s) ;
ajouterFilsDroit(A,s)
```

Etat de la mémoire

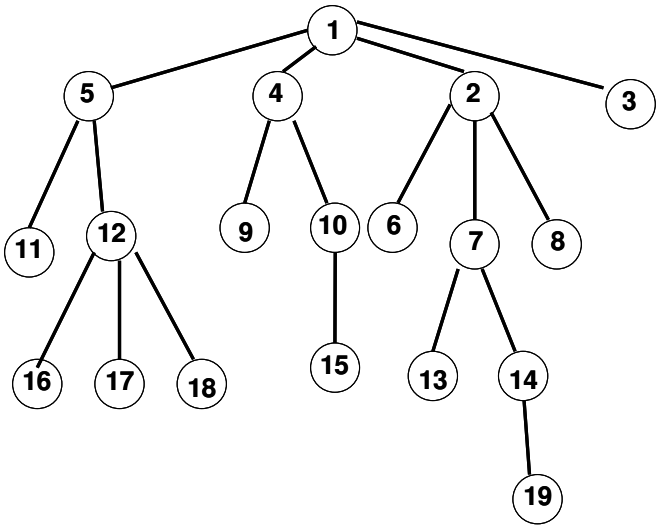


Que remarquez-vous qui peut causer des erreurs?

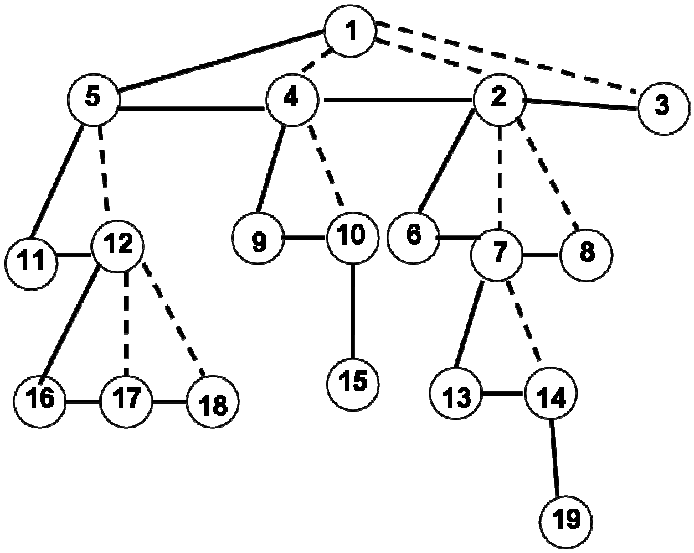
Le fils gauche et le fils droit de la racine partagent la même liste donc sont identiques à la clé près. Sion supprime le fils droit, on sera amené à supprimer la liste associée et ses éléments et le fils gauche ne bougera pas désignant une zone de mémoire devenue inaccessible.

Exercice 2 (5 points)

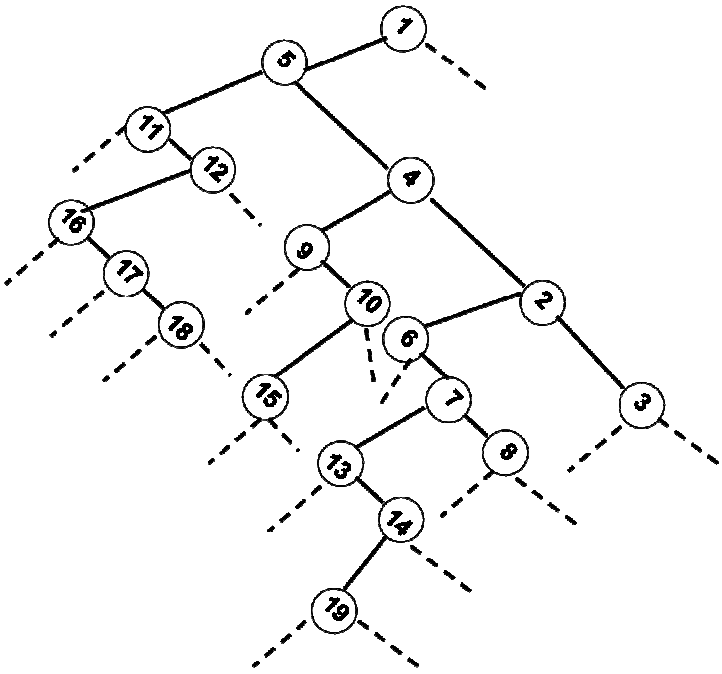
- Soit l'arbre planaire suivant



1. Donnez sa hauteur
   La hauteur est 4.
2. Donnez la liste des sommets dans l'ordre préfixe
   1,5,11,12,16,17,18,4,9,10,15,2,6,7,3,14,19,8,3
3. Donnez la liste des sommets dans l'ordre hiérarchique
   1,5,4,2,3,11,12,9,10,6,7,8,16,17,18,15,13,14,19
4. Donnez l'arbre binaire correspondant dans la bijection de Knuth
   Ajout des liens « frères » et suppression des liens fils



Rotation et ajout de feuilles



5. Pour l'arbre binaire ainsi obtenu, donner la liste des entiers en ordre préfixe, en ordre infixe et en ordre suffixe.
   préfixe : 1,5,11,12,16,17,18,4,9,10,15,2,6,7,13,14,19,8,3
   infixe : 11,16,17,18,12,5,9,15,10,4,6,13,9,14,7,8,2,3,1
   suffixe : 18,17,16,12,11,15,10,9,19,14,13,8,7,6,3,2,4,5,1

Exercice 2 (10 points)

Ecrire une fonction qui renvoie la liste simplement chainée de toutes les valeurs des feuilles d'un arbre binaire d'entier:

1 – En utilisant les primitives du type arbreBinaire et listeSC

```
fonction listeFeuilleRec(val A :arbreBinaired'entier ;ref L :listeSC de entier) :vide ;
  début
    si estFeuille(A) alors
      insérerEnTête(L,valeur(A)
    sinon
     si FilsGauche(A) !=NIL alors
       listeFeuille(FilsGauche(A),L)
     finsi
     si Filsdroit(A) !=NIL alors
       listeFeuille(Filsdroit(A),L)
     finsi
 fin
```

2 – En implémentant directement en allocation dynamique pour le type Arbre Binaire et le type liste simplement chainé.

```
fonction listeFeuille (val A :arbreBinaired'entier ;ref L :listeSC de entier) :vide ;
  var C :curseurListe ;
  début
    si A^.gauche=NIL et A^.droit=NIL alors
      new( c) ;
      c^.info=A^.info ;
      c^.suivant=L.premier;
```

```
      L.premier=c;
    sinon
     si A^.gauche !=NIL alors
      listeFeuille(A^.gauche,L)
     finsi
     si A^.droit !=NIL alors
      listeFeuille(A^.droit,L)
     finsi
fin
```

Listes simplement chainées (listeSC)

```
fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet);
fonction suivant(val L:liste d'objet);
fonction listeVide(val L:liste d'objet): booleen;
fonction créerListe(ref L:liste d'objet):vide;
fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;
fonction insérerEnTete(ref L:liste d'objet  val x:objet):vide;
fonction supprimerAprès(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;
```

Listes doublement chainées (listeDC)

```
fonction finListe(val L:liste d'objet):vide;
fonction précédent(val L::liste d'objet): vide;
```

Piles

```
fonction valeur(ref P:pile de objet):objet;
fonction fileVide(ref P:pile de objet):booléen;
fonction créerPile(P:pile de objet);
fonction empiler(ref P:pile de objet;val x:objet):vide;
fonction dépiler(ref P:pile de objet):vide;
fonction detruirePile(ref P:pile de objet):vide;
```

Files

```
fonction valeur(ref F:file de objet):objet;
fonction fileVide(ref F:file de objet):booléen;
fonction créerFile(F:file de objet);vide;
fonction enfiler(ref F:file de objet;val x:objet):vide;
fonction défiler (ref F:file de objet):vide;
fonction detruireFile(ref F:file de objet):vide;
```

Arbres binaires

```
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;
fonction créerArbreBinaire(val Racine:objet):sommet;
```

Arbres planaires

```
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 créerArborescence(val racine:objet):sommetArbrePlanaire;
fonction ajouterFils(ref S:sommetArbrePlanaire,val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;
fonction créerArbreBPlaniare(val Racine:objet):sommet;
```