Contrôle continu 2007-2008
INF251
Pointeurs - Récursivité – Listes – Piles – Files– Arbres

Questions de cours (5 points)

- Donnez la structure d'un sommet pour un arbre binaire en allocation dynamique.

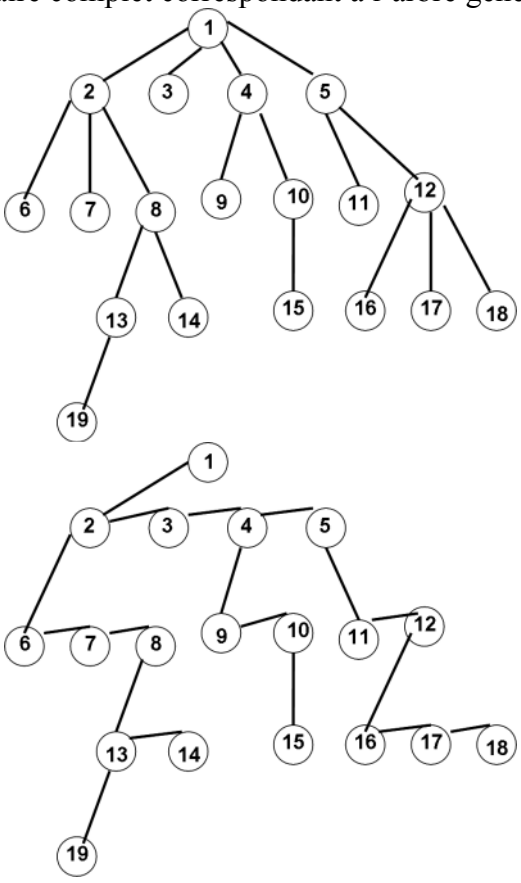
Type sommet=structure

valeur :type_prédéfini ;
gauche :^sommet ;
droit:^sommet ;
pere :^sommet ;

- Un sommet d'un arbre peut-il contenir une chaine de caractère ?

Oui il peut contenir tout type_prédéfini. Cependant, pour des raisons de performance il n'est pas bon qu'un sommet contienne une information de taille importante.

- Donnez l'arbre binaire complet correspondant à l'arbre général suivant

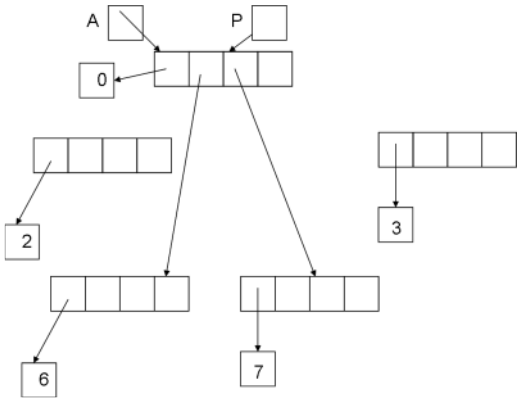


- Dessinez la mémoire, après la suite d'opérations suivante :

```
var A : arbreBinaire de ^entier;
var s :^entier;
var p:^sommet;
new(s) ;
s^=0;
creerArbre(A,s) ;
p=racine(A);
```

```
pour i=1 à 3 pas 2 faire
  new(s) ;
  s^=2*i;
  ajouterFilsGauche(P,s)
  new(s) ;
  s^=2*i+1;
  ajouterFilsDroit(p,s)
   s=p;
 finpour
```

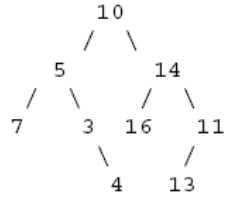
De l'art de faire des trous dans la mémoire !!!!!!!!!!!! Certaines portions seront inaccessibles car non libérées !



Pour vous exercer changez s=p en p=s (dernière instruction de boucle).

Exercice 2 (5 points)

Soit l'arbre suivant :



1 – Donner la liste des entiers en ordre préfixe, en ordre infixe et en ordre suffixe.

Préfixe : 10,5,7,3,4,14,16,11,13

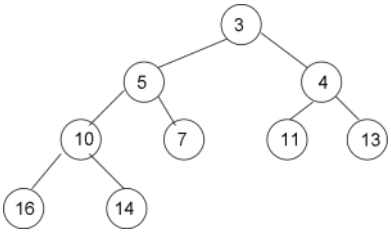
Infixe : 7,5,3,4,10,16,14,13,11

Suffixe : 7,4,3,5,16,13,11,14,10

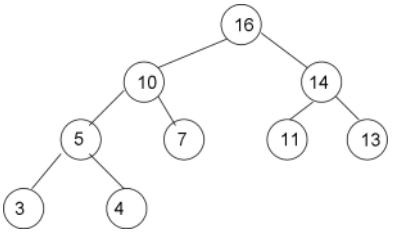
2 – Quelle est la hauteur de cet arbre ?

La hauteur est 3(nombre d'arêtes du plus long chemin de la racine aux feuilles)

3 – Donnez un tas-min contenant les mêmes entiers



4 – Donnez un tas-max contenant les mêmes entiers



Exercice 2 (10 points)

Ecrire une fonction qui renvoie la liste des sommets d'un des chemins de la racine à une feuille de longueur hauteur(A)

1 – En utilisant les primitives du type arbreBinaire

```
Type listeAmelioree=structure
                        L :listeSC de ^sommet ;
                        taille :entier ;
                    finstructure
Fonction chemin(ref A: ^sommet) :listeAméliorée de ^sommet ;
   var U,V :listeAméliorée de ^sommet ;
   Début
      U.taille=0 ;
      Si estFeuille(A) alors
         creerListe(U) ;
         insererEnTete(U.L,A) ;
         retourner(U)
      Sinon
         Si filsGauche(A) !=NIL alors
            U=chemin(filsGauche(A)) ;
         Finsi
         V.taille=0 ;
         Si filsDroit(A) !=NIL alors
            V=chemin(filsDroit(A)) ;
         Finsi
      Finsi
      Si U.taille>V.taille alors
         U.taille=U.taille+1 ;
         insererEnTete(U.L,A) ;
         detruire(V.L)
         retourner(U) ;
      sinon
         V.taille=V.taille+1
         insererEnTete(V.L,A)
         detruire(U.L)
         retourner(V)
      finsi
   fin
fonction detruire(ref L :listeSC de ^sommet) ;
   début
      tantque L !=NIL faire
         supprimerEnTete(L)
      fintantque
   fin
```

2 – En implémentant directement en allocation dynamique.
Il suffit dans ce qui précède de remplacer

estFeuille(A) ***par*** *A^.gauche==NIL et A^.droit==NIL*
filsGauche(A) ***par*** *A^.gauche !=NIL*
filsDroit(A) ***par*** *A^.droit !=NIL*

Listes simplement chainées (listeSC)

fonction premier(val L:type_liste):^type_predefini;
fonction suivant(val L:type_liste; val P:^type_predefini):^type_predefini;
fonction listeVide(val L:type_liste):booléen;
fonction créer liste(ref L:type_liste):vide;
fonction insérerAprès(val x:type_prédéfini;ref L:type_liste; val P:^type_predefini):vide;
fonction insérerEnTete(val x:type_prédéfini;ref L:type_liste):vide;
fonction supprimerAprès(ref L:type_liste;val P:^type_predefini):vide;
fonction supprimerEnTete(ref L:type_liste):vide;

Listes doublement chainées (listeDC), On ajoute les primitives suivantes

fonction dernier(val L:type_liste):^type_predefini;
fonction précédent(val L:type_liste; val P:^type_predefini):^type_predefini;

Piles

fonction valeur(ref P:pile de type_predefini):type_predefini;
fonction pileVide(ref P:pile de type_predefini):booléen;
fonction empiler(ref P:pile de type_predefini; val v:type_predefini):vide;
fonction dépiler (ref P:pile de type_predefini):vide;
fonction créerPile(P:pile de type_predefini);

Files

fonction valeur(ref F:file de type_predefini):type_predefini;
fonction fileVide(ref F:file de type_predefini):booléen;
fonction enfiler(ref F:file de type_predefini; val v:type_predefini):vide;
fonction défiler (ref F:file de type_predefini):vide;
fonction créerFile(F:file de type_predefini);

ArbresBinaire

fonction valeurSommet(val A:arbreBinaire de type_prédéfini, val S:sommet):valeur_prédéfini;
fonction racine(val A:arbreBinaire de type_prédéfini):sommet;
fonction filsGauche(val A:arbreBinaire de type_prédéfini, val S:sommet):sommet;
fonction filsDroit(val A:arbreBinaire de type_prédéfini, val S:sommet):sommet;
fonction père(val A:arbreBinaire de type_prédéfini, val S:sommet):sommet;
fonction créerArbreBinaire(ref A:arbreBinaire de type_prédéfini, val racine:type_prédéfini):vide;
fonction ajouterFilsGauche(val x:type_prédéfini, ref A:arbreBinaire de type_prédéfini,
val S:sommet):vide;
fonction ajouterFilsDroit(val x:type_prédéfini, ref A:arbreBinaire de type_prédéfini,
val S:sommet):vide;
fonction supprimerFilsGauche(ref A:arbreBinaire de type_prédéfini, val S:sommet):vide;
fonction supprimerFilsDroit(ref A:arbreBinaire de type_prédéfini, val S:sommet):vide;

Arbres Binaires en allocation dynamique

fonction valeurSommet(val S:^sommet):valeur_prédéfini;
fonction racine(val A:arbreBinaire de type_prédéfini):^sommet;
fonction filsGauche(val S:^sommet):^sommet;
fonction filsDroit(val S:^sommet):^sommet;
fonction créerArbreBinaire(val racine:type_prédéfini):^sommet;
fonction ajouterFilsGauche(val x:type_prédéfini, val S:^sommet):vide;
fonction ajouterFilsDroit(val x:type_prédéfini, val S:^sommet):vide;
fonction supprimerFilsGauche(val S:^sommet):vide;
fonction supprimerFilsDroit(val S:^sommet):vide;
fonction grefferFilsDroit(ref s1:^sommet;ref s2:arbreBinaire):vide;
fonction grefferFilsGauche(ref s1:^sommet;ref s2:arbreBinaire):vide;
fonction elaguerFilsDroit(ref s1:^sommet):arbreBinaire;
fonction elaguerFilsGauche(ref s1:^sommet):arbreBinaire ;