Les arbres :

I.

1. Le nombre de nœuds :

Type nœud=**structure**

Info : **typeqlq**

FG, FD : **ptr(nœud)**

Fin.

Var Arbre : **ptr (nœud)**

i. Récursif :

FonctionNb_Noeuds (R : **ptr(nœud)**) : **entier**

Debut

SI (R=NIL) alors Nb_Noeuds←0

SINON

Nb_Nœuds←1+Nb_Nœuds(FG(R))+Nb_Nœuds(FD(R))

FSI

FIN.

Commenté [p1]: Nbr de nœud total =nbr de nœud de fils gauche+celle de droite +1 « pour la racine »

ii. Itératif :

Fonction Nb_Nœuds(R : Ptr(Nœud)) : entier

Var Q : Ptr(Nœud)

P : Ptr(Pile)

Nb : entier

Debut

TQ (R<>NIL) ou (Pile_Vide(P) faire

```
        TQ (R <> NIL) faire
            Nb←Nb+1
            EMPILER(P,R)
            R←FG(R)
        FTQ
        DEPILER(P,R)
        P←FD(R)
    FTQ
    Nb_Nœuds←Nb
Fin.
```

2. Nombre de feuilles :

i. Récursif :

```
Fonction Nb_Feuilles(R : ptr(Nœud)) : entier
Debut
    SI (R=NIL) Nb_Feuilles←0
    SINON
        SI (FG(R)=NIL) et (FD(R)=NIL) alors Nb_feuilles←1
        SINON
        Nb_feuilles←Nb_feuilles(FG(R))+Nb_feuilles(FD(R))
        FSI
    FSI
Fin.
```

ii. Itératif :

```
Fonction Nb_Feuilles(R : ptr(nœud) : entier
```

```
Var Nb : entier
Debut
   Créer_pile(P)
   Nb←0
   TQ (R<>NIL) ou (>Pile-vide(p)) faire
      TQ (R<> NIL) faire
         SI (FD(R)=NIL) et (FG(R)=NIL) alors Nb←Nb+1
         EMPILER(P,R)
         R←FG(R)
      FTQ
      DEPILER(P,R)
      R←FD(R)
   FTQ
   Nb-feuilles←Nb
Fin.
```

3. Somme des contenus de tous les nœuds :

i. Récursif :

```
Fonction Somme_Nœuds(R : ptr(Nœud)) : entier
Debut
   Si (R=NIL) Somme_Noeuds←0
   SINON
      Somme_noeuds←Info(R)+Somme_Noeuds(FG(R))+Som
      me_Noeuds(FD(R))
   FSI
Fin.
```

ii. Itératif :

```
Fonction Somme_noeuds(R : Ptr(Nœud)) : entier
Var Som : entier
Debut
   Créer_Pile(P)
   Som←0
   TQ (R<>NIL) ou (>Pile-Vide(P)) faire
      TQ (R<>Nil) Faire
         Som←Som+Info(R)
         EMPILER(P,R)
         R←FG(R)
      FTQ
      DEPILER(P,R)
      R←FD(R)
   FTQ
   Somme_Noeuds←Som
Fin.
```

4. La profondeur :

i. Récursif :

```
Fonction Profondeur(R : Ptr(Nœud)) : entier
Debut
   Si (R=Nil) Profondeur←-1
   SINON
      Profondeur←1+Max(Profondeur(FG(R),Profondeur(FD(R
      )))
```

FSI

Fin.

ii. Itératif :

```
Fonction Profondeur (R : ptr(Nœud)) : entier
Var Prof, max : entier
Debut
   Créer_Pile(P)
   Prof←0
   Max←-1
   TQ (R<>Nil) ou(>Pile-vide(p)) faire
      Prof←prof-1
      TQ (R<>Nil) faire
         Prof←Prof+1
         EMPIlER(P,R)
         R←FG(R)
      FTQ
      Si (Prof>Max) alors max←prof
      DEPILER(P,R)
      R←FD(R)
   FTQ
   Profondeur←Max
Fin.
```

II.

1. Arbre strictement binaire :

i. Récursif :

Fonction bin_Strict(R : ptr(nœud)) : booléen
Debut
Si (R=Nil) alorsBin_Strict←Vrai
Sinon
Si [(FG(R)=Nil)et(FD(R)<>Nil)] ou [(FG(R)<>Nil) et (FD(R)=Nil) alorsBin_Strict←faux
Sinon
Bin_Strict←[Bin_strict(FG(R))]et[Bin_strict[FD(R)]
FSI
FSI
Fin.

2. Complet :

i. //elle n'est pas récursive car elle appelle une autre fonction//

Fonction Complet (R : Ptr(nœud)) : Booléen
Début
Si (R=Nil) alors Complet←vrai
Sinon
Si (Bin_Strict(R) et memelvl(R)) alors complet←vrai
Sinon Complet←faux
FSI
FSI
Fin.

ii.

Fonction Arb_Complet(R :ptr(Nœud)) :Booléen

```
Debut
   Si (R=Nil) alors Arb_Complet←vrai
   Sinon
      Si [(Profondeur(FG(R))<>Profondeur(FD(R))] ou
      [(Strict_bin(FG(R)) et (>Strict_bin(FD(R)) ou
      (>strict_bin(FG(R)) et (Strict_bin(FD(R))] alors
         Arb_Complet←Faux
      Sinon
         Arb_complet←(arb_complet(FG(R)) et
         (Arb_Complet(FD(R))
      FSI
   FSI
Fin.
```

III. Trouver les doubles dans une suite de n nombres :

IV. Algorithme de parcours :

1. Pré ordre :

i. Récursif :

```
Procedure Preordre(R :Ptr(nœud))
Debut
   Si (R<>Nil) alors
      Ecrire(Info(R))
      Preordre(FG(R))
      Preordre(FD(R))
   FSI
Fin.
```

ii. Itératif :

```
Procedure Preordre(R :ptr(Nœud))
Debut
   Créer_pile(p)
   TQ (R<>Nil) ou (>Pile_Vide(p))
      TQ (R<>Nil)
         Ecrire(info(R))o
         Si (FD(R)<>Nil)
            Empiler(p,FD(R))
         FSI
         R←FG(R)
      FTQ
      Depiler(P,R)
   FTQ
Fin.
```

2. Innordre :

i. Récursif :

```
Procedure Inordre(R :ptr(Nœud))
Debut
   Si (R<>Nil) alors
      Inordre(Fg(R))
      Ecrire(Info(n))
      Inordre(FD(R))
   FSI
Fin.
```

ii. Itératif :

```
Procedure Inordre(R :ptr(Nœud))
Debut
   Créer_pile(p)
   TQ (R<>Nil) ou (>Pile_Vide(p))
      TQ (R<>Nil) faire
         Empiler(p,R)
         R←FD(R)
      FTQ
      Depiler(R,p)
      Ecrire(Info(R))
      R←FD(R)
   FTQ
Fin.
```

3. Post ordre :

i. Récursif :

```
Procedure Postordre(R :ptr(Nœud))
Debut
   Si (R<>Nil) alors
      Postordre(Fg(R))
      Ecrire(Info(n))
      Postordre(FD(R))
   FSI
Fin.
```

ii. Itératif :

```
Type Sauv=Structure
                Nœud : ptr(nœud)
                Indice : booléen
Fin.
Procedure Post_Ordre(R :ptr(nœud))
Debut
   TQ (R<>Nil) ou (>PileVide(P) faire
      TQ (R<>Nil) faire
         Sauv.Nœud←R
         Sauv.indice←Vrai
         Empiler(P,Sauv)
         R←FG(R)
      FTQ
      Depiler(P,Sauv)
      R←Sauv.Nœud
      Si (Sauv.indice) alors
         Sauv.indice←faux
         Empiler(p,sauv)
         R←FD(R)
      Sinon
         Ecrire(Info(R))
         R← Nil
      FSI
   FTQ
Fin.
```

V. Recherche, Insertion et Suppression :

1. Recherche :

```
Procedure Recherche (R : ptr(nœud), var Q :ptr(nœud), var père : ptr(nœud), var Trouv : Booléen)
```

```
Var P : ptr(nœud)
Debut
Trouv←faux
P←R
Père←NIL
Q←NIL
TQ (R<>NIL) et (>Trouv)
   SI (Info(p)=v) alors
      Trouv←vrai
      Q←P
   Sinon
      père←P
      Si (Info(P)>v) alors
         P←FG(P)
      Sinon
         P←FD(P)
      FSI
   FSI
FTQ
Fin.
```

2. Insertion :

```
Fonction Insertion (R : Ptr(nœud), V : entier) : Ptr(Nœud)
Var P, Père : Ptr (nœud)
      Trouver : Booléen
Debut
```

```
Recherche(R, P, Père, Trouver, v)
Si (Trouver=vrai) ou (P<>NIL) alors Ecrire ('Insertion impossible')
Sinon
    Créer_ Nœud (P)
    Aff_ Info (P,v)
    Si (Père=NIL) alors R←P//L'arbre n'existe pas
    Sinon
        Si (V<Info(Père)) alors Aff_ FG (Père, P)
        Sinon Aff_ FD (Père, P)
    FSI
FSI
Fin.
```

3. Suppression :

```
Procédure Supprimer (R : Ptr(nœud), Q :Ptr(Nœud)
Debut
    Recherche (R, P, Per, Trouv, Val)
    Si (P=NIL) alors Ecrire ('Suppression Impossible')
    Sinon
        Si (FG(P)=NIL) alors
            Si (FD(P)=NIL) alors //Cas n°1
                Chaînage (R, Père, P, Val, NIL)
            Sinon //Cas n°2
                Chaînage (R, Père, P, Val, FD(P))
            FSI
```

```
        Sinon
            Si (FD(P)=NIL) alors //Cas n°3
                Chaînage (R, Père, P, Val, FG(P))
            Sinon //Cas n°4
                Cas_Quatre (P)
            FSI
        FSI
    FSI
Fin.

Procédure Chaînage (Var R :ptr, Père :ptr, P :ptr, val : entier,
Q :ptr)
Debut
    Si (Père=NIL) alors R←Q
    Sinon
        Si (Val>Info(Père)) alors Aff_FD (Père, Q)
        Sinon Aff_FG (Père, Q)
    FSI
    Liberer (P)
Fin.

Procédure Plus_ Petit (Var PElt :ptr, Var Elt :entier, Var
Per_elt : ptr)//Utilisée pour le quatrième cas
Début
    TQ (FG (Pelt) <> NIL) faire
        Père_elt←PElt
        PElt←FG (PElt)
```

```
    FTQ
    Elt←Info(PElt)
Fin.

Procédure Cas_Quatre (P : ptr)
Var Q, PèreQ : ptr
        Val : entier
Début
    Q←FD(P)
    PèreQ←P
    Plus_Petit(Q, Val, PèreQ)
    Aff_Info (P, Val)
    Si (PèreQ=P) alors Aff_FD (PèreQ,FD(Q))
    Sinon Aff_FG (PèreQ, FD(Q))
    Libérer (Q)
Fin.
```