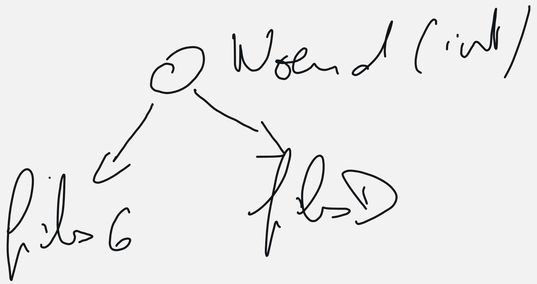
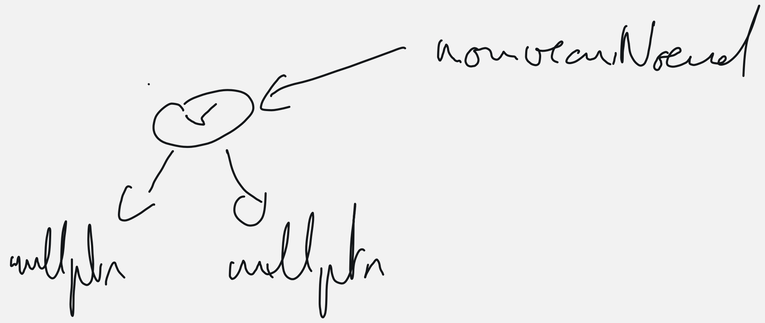
1 Manipulation d'arbres binaires

- Définir une structure de données permettant de manipuler un arbre binaire d'entiers.
- Écrire les fonctions permettant :
  - le calcul du nombre de nœuds de l'arbre,
  - le calcul du nombre de feuilles de l'arbre,
  - le calcul de la hauteur de l'arbre,
  - la somme des valeurs des nœuds de l'arbre,
  - l'affichage infixe et postfixe des valeurs des nœuds de l'arbre.



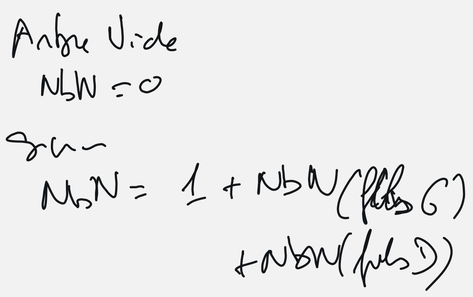

```
struct Noeud{
   int valeur;
   Noeud* filsG;
   Noeud * filsD;
};
```

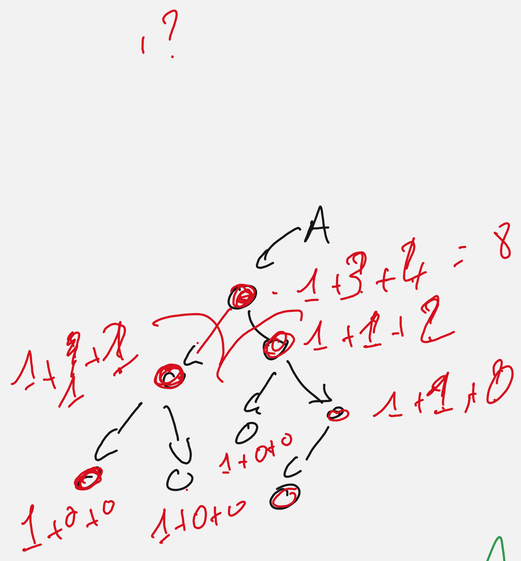
```
Noeud * creerNoeud(int v)
{
   Noeud * nouveauNoeud = new Noeud;
   nouveauNoeud -> valeur = v;
   nouveauNoeud -> filsG = nullptr;
   nouveauNoeud -> filsD = nullptr;
   return nouveauNoeud;
}
```



Error loading webview: Error: Could not register service worker: InvalidStateError: Failed to register a ServiceWorker: The document is in an invalid state..

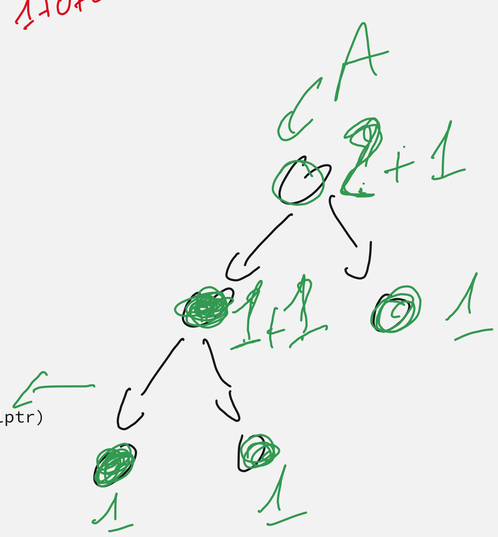
le calcul du nombre de nœuds de l'arbre,






```
int NbNoeuds( Noeud *A){

    if(A == nullptr){
        return 0;
    }else{
        return 1+ NbNoeuds(A->filsG) +
NbNoeuds(A->filsD);
    }
}
```

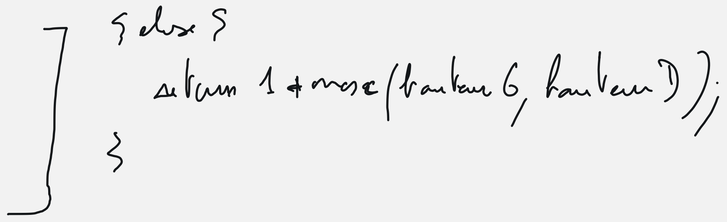
le calcul du nombre de feuilles de l'arbre,

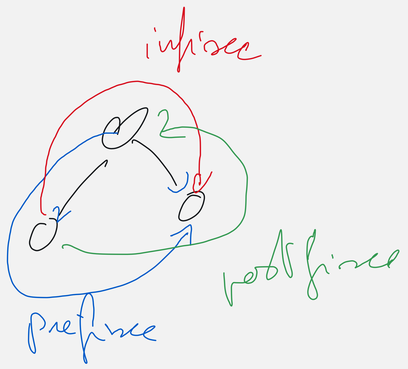



```
int nbFeuilleArbre(Noeud * sourceArbre){
    if (sourceArbre == nullptr)
    {
        //Si l'arbre est vide
        return 0;
    }
    else{
        if ( (sourceArbre->filsD == nullptr && sourceArbre->filsG == nullptr)
)
        {
            return 1;
        }else{
            return  nbFeuilleArbre(sourceArbre->filsD) +
nbFeuilleArbre(sourceAbre->filsG);
        }
    }
}
```

le calcul de la hauteur de l'arbre,

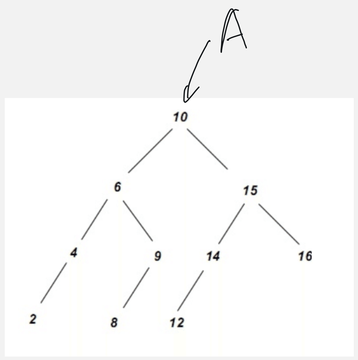
```
        int hauteur(Noeud* arbre){
            if(arbre == nullptr){
                return 0;
            }

            int hauteurG =
        hauteur(arbre->filsG);
            int hauteurD =
        hauteur(arbre->filsD);

            if(hauteurG > hauteurD){
                return hauteurG + 1;
            }else {
                return hauteurD + 1;
            }
        }
```





la somme des valeurs des nœuds de l'arbre,

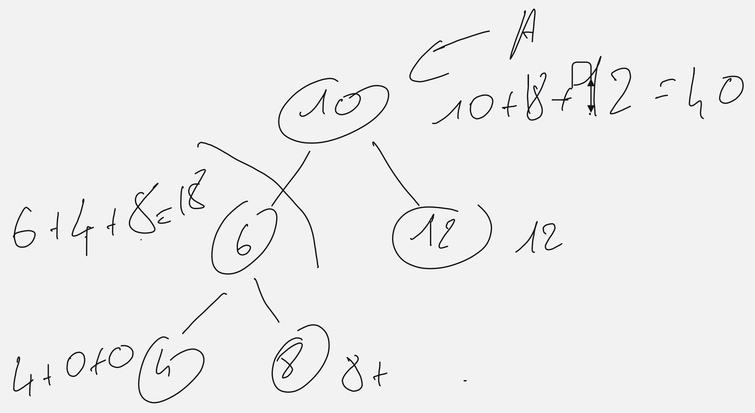
```
int sommeNoeud(Noeud* abr){
          if (abr==nullptr){
            return 0;
        }
       else {
        return abr->valeur+sommeNoeud(abr->filsG)
+sommeNoeud(abr->filsD);
      }
}
```



prefixe : 10 6 4 2 9 8 15 14 12 16

infixe : 2 4 6 8 9 10 12 14 15 16

postfixe : 2 4 8 9 6 12 14 16 15 10




```
void prefixe (Noeud * A )
{   if (A!= nullptr)
    {  cout << A -> valeur;
       prefixe (A -> fils G);
       prefixe (A -> fils D);
    }
}
```

```
void infixe (Noeud * A )
{   if (A!= nullptr)
    {
       infixe (A -> fils G);
       cout << A -> valeur;
       infixe (A -> fils D);
    }
}
```

```
void postfixe (Noeud * A )
{   if (A!= nullptr)
    {
       postfixe (A -> fils G);
       postfixe (A -> fils D);
       cout << A -> valeur;
    }
}
```

2 Manipulation d'arbres binaires de recherche

Un arbre binaire de recherche est un arbre binaire dont les étiquettes des nœuds appartiennent à un ensemble totalement ordonné et qui vérifie :

- pour tout nœud x détiquette e, pour toute étiquette g de la branche gauche de x, $g \leq e$,
- pour tout nœud x détiquette e, pour toute étiquette d de la branche droite de x, $d \geq e$.

Écrire les fonctions permettant :

- de vérifier qu'un arbre binaire est un arbre binaire de recherche
- de tester si une valeur est dans un arbre binaire de recherche
- de créer un élément de l'arbre
- d'ajouter un élément dans l'arbre,
- de supprimer un élément de l'arbre

-> supérieur
inférieur

bool sup(Noeud *A, int v)

Renvoie vrai si v passé en paramètre en sup ou = à toutes les valeurs de noeuds de l'arbre

```
bool sup(Noeud* A, int v){
    if(A == nullptr){
        return true;
    }

    if(A->valeur >= v){
        return sup(A->filsG, v) && sup(A->filsD, v);
    }

    return false;
}
```

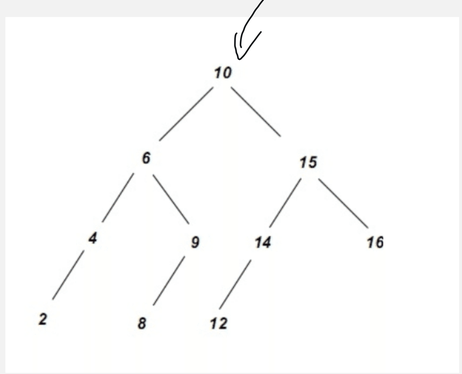
bool inf(Noeud *A, int v)

Renvoie vrai si v passé en paramètre en inf ou = à toutes les valeurs de noeuds de l'arbre

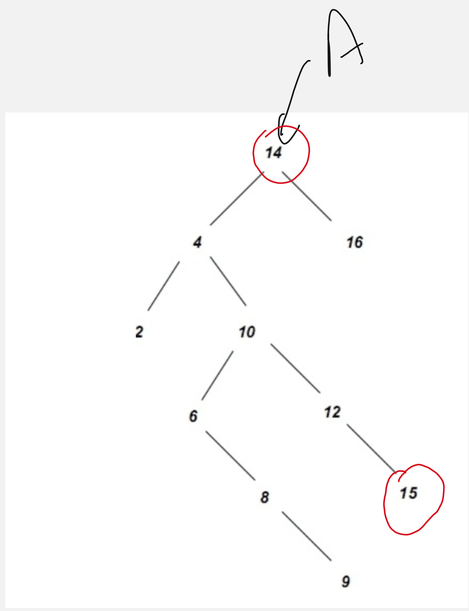
```
bool inf(Noeud* A, int v){
    if(A == nullptr){
        return true;
    }else{


    return (A->valeur =< v) && inf(A->filsG, v) && inf(A->filsD, v) ;

}


}
```

```
bool estABR(Noeud* A)
{
    bool ok = true;
    if(A != nullptr){
        ok = inf(A->filsD, A->valeur) && sup(A->filG, A->valeur)
            && estABR(A->filsG) && estABR(A->filsD);
    }
    return ok;
}
```

A



A




```
void detruireNoeud(Noeud *A)
{
    if(A != nullptr){
        detruireNoeud(A->filG);
        detruireNoeud(A->filsD);
        delete A;
    }
}
```

2 Manipulation d'arbres binaires de recherche

Un arbre binaire de recherche est un arbre binaire dont les étiquettes des nœuds appartiennent à un ensemble totalement ordonné et qui vérifie :

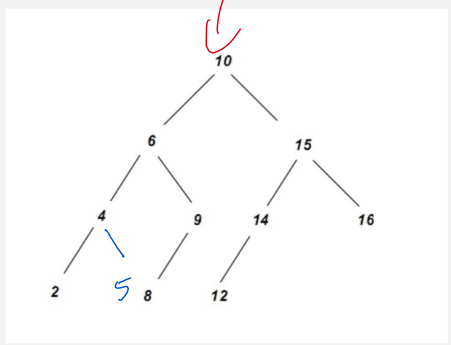
— pour tout nœud x détiquette e, pour toute étiquette g de la branche gauche de x, $g \leq e$,
— pour tout nœud x détiquette e, pour toute étiquette d de la branche droite de x, $d \geq e$.

Écrire les fonctions permettant :

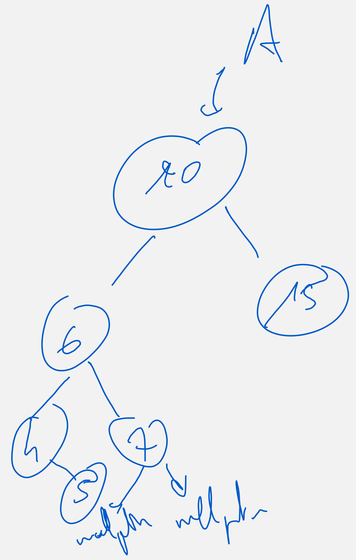
— de vérifier qu'un arbre binaire est un arbre binaire de recherche
— de tester si une valeur est dans un arbre binaire de recherche
— de créer un élément de l'arbre
— d'ajouter un élément dans l'arbre,
— de supprimer un élément de l'arbre

de tester si une valeur est dans un arbre binaire de recherche

```
bool estPresenteABR(Noeud * A, int v)
{
        bool ok = false;
        if(A != nullptr){
                  if(A->valeur > v)
                 {
                   ok = estPresenteABR(A->filsG, v);
                 }
                 else if(A->valeur < v) {
                        ok = estPresenteABR(A->filsD, v);

                }else{ // égalité
                        ok = (A->valeur == v);
                }
        }
        return ok;
}
```




```
Noeud*  ajout(Noeud*  A, int v){
   if(A == nullptr)
      {
        return creerNoeud(v);
      }else{
         if (estPresenteABR(A, v))
         {
           return A;
         }else{
            if (A->valeur > v)
            {
                A->filsG = ajout(A->filsG, v);
             }else{ //v est plus grand que la valeur dans A
                A->filsD = ajout(A->filsD,v);
             }
          return A;
      }
   }
```



```
Noeud* supprimer(Noeud *A, int v){
   if(A !=nullptr){
      if(estPresentABR(A,v)){
         if (v < A->valeur){
            A->filsG = supprimer(A->filsG, v);
         } else if(v > A->valeur){
            A->filsD = supprimer(A->filsD,v);
         }else{ // v= A->valeur
            // si l'élément à supprimer est une feuille
            if(A->filsD == nullptr && A-> filsG== nullptr){
               detruireNoeud(A);
            }else if(A->filsD == nullptr || A-> filsG== nullptr) {// noeud  sup a un seul fils

               Noeud * temp = A;
               if(A->filsG != nullptr)
                {
                    A = A->filsG;
                }
                else {
                   A = A->filsD;
                }
               detruire(temp);
            }else{ //noeud à sup a 2 fils
               Noeud * tmp = minFilsD(A);
               Noeud *tmp2 = A;
               A->filsD = supprimer(A->filsD, tmp->valeur);
               tmp->filsG = A->filsG;
               tmp->filsD = A->filsD;
               A= tmp;
               detruire(tmp2);
            }
      }
   }
   return A;
}
```

```
Noeud * minFilsD(Noeud*A){ // Noeud le plus à G du fils D dans un ABR
   Noeud *N = nullptr;
   if( A != nullptr){
      if(A->filsD != nullptr){
         N= A->filsD;
         while(N->filsG != nullptr){
            N= N->filsG;
         }
       )
   }
   return N;
}
```

6.2 Réservation de salle

Une salle de réception peut être réservée par journée, sur un intervalle de jours donnés $I = [debut, fin]$. On représente les réservations sous forme d'un arbre binaire de recherche pour accélérer la recherche, l'ajout ou la suppression d'une réservation.

— Proposez une structure de données pour stocker les réservations,

— Écrivez l'algorithme principal permettant au gestionnaire de la salle de vérifier si une réservation peut être ajoutée, de l'ajouter si possible ou de supprimer une réservation, et d'afficher les réservations dans l'ordre chronologique,

— Écrivez les algorithmes et spécifications des fonctions utilisées.