```
#include <iostream>
#include<stdlib.h>

/* run this program using the console pauser or add your own getch, system("pause") or input loop */

typedef struct noeud {
	int valeur;
	struct noeud *filsg;
	struct noeud *filsd;
}*Arbre ;

void parcoursInfixe(Arbre ar) {
		if(ar!=NULL){
			printf("%d ", ar->valeur); //traitement
			parcoursInfixe(ar->filsg);
			parcoursInfixe(ar->filsd); // appel récursive
		}

}

Arbre preparerNoeud(int e) {
   Arbre n =(Arbre)malloc(sizeof(noeud));
   n->valeur = e;
   n->filsg = NULL;
   n->filsd = NULL;
  return n;
}

Arbre nouveau_noeud(int a){
  Arbre x;
  x =(Arbre) malloc(sizeof(noeud));
  if (!x) {
    perror("Echec malloc");
  }

  x->filsg = NULL;
  x->filsd = NULL;
  x->valeur = a;
  return x;
}

int comparer(int a, int b){
  if (a < b) return 1;
  if (a > b) return -1;
  return 0;
}


void inserer(Arbre *px, int a){
  if (!(*px)) {
    *px = nouveau_noeud(a);
  }
  else {
    Arbre y, parent;
    int cote;
    y = *px;
```

```
    // Recherche de la place de a (parent et côté où le mettre)
    while (y) {
      parent = y;
      cote = comparer(a, y->valeur);
      switch(cote){
      case 1: // côté gauche
        y = y->filsg;
        break;
      case -1: //côté droit
        y = y->filsd;
        break;
      case 0: // on a déjà cet élément, pas d'insertion
        return;
      }
    }
    // parent sera le parent du nouveau noeud, la valeur de cote dit de quel côté l'attacher à son parent.
    y = nouveau_noeud(a);

    if (1 == cote) {
      parent->filsg = y;
    }
    else {
      parent->filsd = y;
    }
  }
}

void insererRecurssive(Arbre *A, int a){
            if (!(*A)) {
              *A = nouveau_noeud(a);
            }
            else if(a<(*A)->valeur)
                        insererRecurssive(&((*A)->filsg),a);
            else
                        insererRecurssive(&((*A)->filsd),a);

}
/*void parcours_infixe(Arbre *x){
  if (!(*x)) {
    parcours_infixe(&((*x)->filsg));
    printf("%d ",(*x)->valeur);
    parcours_infixe(&((*x)->filsd));
  }
}*/

void parcours_infixe(Arbre x){
  if (x) {
            printf("{");
    parcours_infixe(x->filsg);
    printf(",%d,",x->valeur);
    parcours_infixe(x->filsd);
    printf("}");
  }
}

int cardinal(Arbre x){
```

```
 if (x) {
   return cardinal(x->filsg) + cardinal(x->filsd) + 1;
 }
 else return 0;
}



void initialiser(Arbre*A){
        A=NULL;
}

void Remplacer(Arbre *q) {
        Arbre*R, *S;
        *R=(*q)->filsg;
        S=R;
        while  ((*R)->filsd !=NULL) {
           S=R;
           *R=(*R)->filsd;
        }
        (*q)->valeur = (*R)->valeur;
        printf("trouve ");
        if ((*R)->filsg !=NULL)
           (*S)->filsd= (*R)->filsg;
}

void Supprimer(Arbre *p, int x) { //recherche du nœud à supprimer
        Arbre *q;
        if (x< (*p)-> valeur) Supprimer(&((*p)->filsg),x);
        else if (x> (*p)->valeur) Supprimer(&((*p)->filsd),x);
             else  { // p pointe vers le nœud à supprimer
                  q=p;
                  if (&((*q)->filsd) == NULL)  *p=(*q)->filsg;
                  else if (&((*q)->filsg) == NULL)  *p=(*q)->filsd;
                       else Remplacer((q)); // deux fils
                  free(q);
               }
}
int max(int a, int b){
        if (a>b) return a;
        return b;
}
int hauteur(Arbre x){
 if (x) {
   return 1 + max(hauteur(x->filsg), hauteur(x->filsd));
 }
 else return 0;
}

void niveau(Arbre a, int n){
        if(a!=NULL){
                if(n==0) printf(" %d ",a->valeur);
                else{
                        niveau(a->filsg,n-1);
                        niveau(a->filsd,n-1);
                }
```

```
	}

}

int noeudNiveau(Arbre a, int n){
	if(a!=NULL){
		if(n==0) return 1;
		else{
			int n1=noeudNiveau(a->filsg,n-1);
			int n2=noeudNiveau(a->filsd,n-1);
			return n1+n2;
		}
	}

}

void supprimerArbreRef(Arbre &A){
	if(A!=NULL){
		supprimerArbreRef(A->filsg);
		supprimerArbreRef(A->filsd);
		free(A);
		A=NULL;
	}
}

void supprimerArbre(Arbre* A){
	if(A!=NULL){
		Arbre* temp=&(*A)->filsg;
		supprimerArbre(temp);
		supprimerArbre(temp);
		free(A);
		A=NULL;

	}
}
int supprimerNoeudAvecDessendantRef(Arbre&A, int x){
	if(A!=NULL){
		if(A->valeur==x) {
			supprimerArbreRef(A);
			return 1;
		}
		else if(! supprimerNoeudAvecDessendantRef(A->filsg,x))
			return supprimerNoeudAvecDessendantRef(A->filsd,x);
		else return 1;

	}
}

int supprimerNoeudAvecDessendant(Arbre*A, int x){
	if(A!=NULL){
		if((*A)->valeur==x) {
			supprimerArbreRef(*A);
			return 1;
		}
		else if(! supprimerNoeudAvecDessendant(&((*A)->filsg),x))
			return supprimerNoeudAvecDessendant(&((*A)->filsd),x);
```

```
                else return 1;

        }
}

void afficherFeuille(Arbre a){
        if(a!=NULL){
                if(a->filsg==NULL && a->filsd==NULL) printf("%d ",a->valeur);
                else{
                        afficherFeuille(a->filsg);
                        afficherFeuille(a->filsd);
                }
        }
}

void afficherNoeudsInternes(Arbre a){
        if(a!=NULL){
                if(a->filsg!=NULL || a->filsd!=NULL) printf("%d ",a->valeur);
                if (a->filsg!=NULL)
                        afficherNoeudsInternes(a->filsg);
                if (a->filsd!=NULL)
                        afficherNoeudsInternes(a->filsd);

        }
}

void tousNiveau(Arbre A){
        int n=hauteur(A);
        int i=0;
        while(i<n){
                niveau(A,i);
                i++;
        }

}
void Remplacer(Arbre &q) {
        Arbre R, S;

        R=q->filsg;
        S=R;
        while  (R->filsd !=NULL) {
           S=R;
           R=R->filsd;
           printf("<<%d>>",R->valeur);
        }
        q->valeur=R->valeur;
        printf("<%d> ",   q->valeur );
        if (R->filsg !=NULL)
          S->filsd= R->filsg;
}

supprimerNoeud(Arbre &p, int x) { //recherche //du nœud à supprimer
        Arbre q;
        if (x< p->valeur) supprimerNoeud(p->filsg,x);
        else if (x> p->valeur) supprimerNoeud(p->filsd,x);
        else  { // p pointe vers le nœud à
```

```
                //supprimer
                q=p;
                if (q->filsd == NULL)  p=q->filsg;
                else if (q->filsg == NULL)  p=q->filsd;
                else Remplacer(q); // deux fils
                free(q);
            }
}

Arbre supprimerElement(Arbre & a, int val)
{
   Arbre tmp;
   if( a->valeur == val )
   {
     if( a->filsg != NULL)
     {
       // on accroche a->fd au fils le plus à droite du fils gauche
       tmp = a->filsg;
       while(tmp->filsd != NULL)
       {
         tmp = tmp->filsd;
       }
       tmp->filsd = a->filsd;
       a = a->filsg;
     }
     else
     {
        a = a->filsd;
     }
   }
   else
   {
      if( val < a->valeur )
      {
         a->filsg = supprimerElement( a->filsg, val);
      }
      else
      {
         a->filsd = supprimerElement( a->filsd, val);
      }
   }
   return a;
}

int main(int argc, char** argv){
            Arbre           x = NULL;
            //Arbre x;
            //initialiser(&x);
   int      i;
   int      tab[] = {5, 2, 4, 3, 7, 6, 1, 9, 8, 0};
   int      tab2[] = {1, 4, 5, 3, 2};

   for (i = 0; i < 10; i++) {
    inserer(&x, tab[i]);
    //insererRecurssive(&x, tab[i]);
   }
```

```
    parcours_infixe(x);
    printf("\n feulles:\n");
    afficherFeuille(x);
    printf("\n noeud internes:\n");
    afficherNoeudsInternes(x);
    printf("\n---------supprimer noeud---------\n");
    supprimerElement(x,7);
   // supprimerNoeud(x,7);
     printf("\n Taille:%d \n",cardinal(x));
    parcours_infixe(x);
     printf("\n feulles:\n");
    afficherFeuille(x);
    printf("\n fin infixe\n");
   // printf("\n Taille:%d \n",cardinal(x));


    printf("\n fin infixe\n");
    printf("\n Taille:%d \n",cardinal(x));
            printf("\n Hauteur:%d \n",hauteur(x));
             printf("\naffichage des niveau\n");
             printf("\nombre noeud niveau:%d",noeudNiveau(x,1));

    supprimerArbreRef(x);
     printf("\n Taille:%d \n",cardinal(x));
    parcours_infixe(x);

    supprimerNoeudAvecDessendant(&x,5);
    printf("\n Taille:%d \n",cardinal(x));
    parcours_infixe(x);


            return 0;
}
```