PARTIE VII : LES ARBRES BINAIRES EN C

- **Implémentation**

```
#includ <stdlib.h>
Typedef   struct   node  {
                              Item   info;
                              Struct  node   *gauche;
                              Struct   node   *droit;
                           }   node;
Typedef   node   *arbre;

Arbre   A = NULL;       /*A étant un arbre*/
```

NB : Pour un arbre de recherche « info » est appelée « clé » et son type est unsigned Int. Donc « item » sera remplacé par « unsigned Int ».

- **On essaie d'écrire les trois procédures de parcours : préfixe, infixe et postfixe.**
  - **Procédure préfixe.**

```
Void visiterPrefixe(arbre A)
{
        Printf(''clé = %\n'', A->info) ;
        If (A->gauche != NULL)
           Visiterprefixe(A->gauche) ;
        If (A->droit != NULL)
           visiterPrefixe(A->droit) ;
}
```

  - **Procédure postfixe.**

```
Void visiterPostfixe(arbre A)
{
        If (A->gauche != NULL)
           VisiterPostfixe(A->gauche) ;
        If (A->droit != NULL)
           visiterPostfixe(A->droit) ;
        Printf(''clé = %\n'', A->info) ;
}
```

- **Procédure infixe.**

```
Void visiterIntfixe(arbre A)
{
        If (A->gauche != NULL)
           VisiterInfixe(A->gauche) ;
        Printf(''clé = %\n'', A->info) ;
        If (A->droit != NULL)
           visiterInfixe(A->droit) ;
}
```

- **Procédure d'ajout d'un nœud dans un arbre de recherche.**

```
Void  addnode(arbre  **A, unsigned Int cle)
{
        Arbre *Ptnode ;   /*Pointeur temporaire pour les noeuds*/
        Arbre *Ptarbre ;  /*Pointeur temporaire pour arbre er sous arbres*/
        Arbre *nouveaunode = malloc(Sizeof(arbre)) ;
        Nouveaunode->info = cle ;
        Nouveaunode->gauche = NULL ;
        Nouveaunode->droit = NULL ;
        If  (Ptarbre != NULL)
         Do
            {
                Ptnode = Ptarbre ;
                If (cle > ptarbre->info)
                 {
                    Ptarbre = Ptarbre->droit ;
                    If (Ptarbre = NULL)
                        Ptnode->droit = nouveaunode ;
                    }
                Else
                    {
                        Ptarbre = Ptarbre->gauche ;
                        If (Ptarbre = NULL)
                            Ptnode->gauche = nouveaunode ;
                     }
                }

            While (Ptarbre != Null) ;
        Else /* L'arbre est vide donc il sera constitué d'un seul nœud : racine*/
            *A = nouveaunode ;
}
```

- **Recherche dans un arbre : On retourne la valeur 1 si la clé est trouvée et 0 sinon.**

```
Int rechercheNode(arbre *A, unsigned cle)
{
        While (A != NULL)
        {
           If (cle == A*info)
             Return 1 ;
           If (cle > A->info)
              A = A->droit ;          /* Explorer le sous arbre droit*/
           Else
              A = A->gauche ;         /* Explorer le sous arbre gauche*/
        }
        Return 0 ;
}
```

- **Afficher le contenu d'un arbre.**

```
Void  afficherArbre(arbre *A)
{
   If  (A = NULL)
      Return ;
   If (A->gauche != NULL)
      AfficherArbre(A->gauche);
   printf(''Clé = %\n'', A->info);
   if  (A->droit != Null)
      AfficherArbre(A->droit) ;
}
```

- **Détruire un arbre (Effacer).**

```
Void EffacerAbre(arbre **A)
{
   Arbre  *Ptarbre = *A ;
   If (A = NULL)
      Return ;
   If (Ptarbre->gauche != NULL)
      EffacerArbre(Ptarbre->gauche);
   Free(Ptarbre) ;
   If (Ptarbre->droit != NULL)
      EffacerArbre(Ptarbre->droit) ;
   *A = NULL ;
}
```