**Exercice 1 :**

```
struct nœud{
int valeur;
struct nœud *suivant;
};
typedef struct nœud nœud;
struct list{
struct nœud *tete;
};
typedef struct list list;
// Question 1
nœud *rechercher(list l, int v){
nœud* resultat=l.tete;
if(l.tete==NLL)
return NULL;
else{
        while(resultat!=NULL){
                if(resultat->valeur==v)
                return resultat;
                resultat=resultat->suivant;
        }
}
}
// Question 2
int somme(liste *l){
        int s=0;
        while(l->tete!=NULL){
                s+=l->tete->valeur; // s=s+l->tete->valeur;
                l->tete=l->tete->suivant;
```

```
        }
        return s;
}
//Question 3
int min(liste *l){
        int min=l->tete->valeur;
        l->tete=l->tete->suivant;
        while(l->tete!=NULL){
                if(l->tete->valeur<min))
                min=l->tete->valeur;
                l->tete=l->tete->suivant;
        }
        return min;
}
// Question 4
bool existe(liste *l, int x){
        while(l->tete!=NULL && l->tete->valeur!=x){
                l->tete=l->tete->suivant;
        }
        if(l->tete==NULL)
                return false;
                else return true;
}
// Question 5
void supprimerValeur(list *l, int v){
        nœud* courant=l->tete;
        nœud* precourant->l->tete; // utilisé pour conserver l'élément précedent
        while(courant!=NULL){
                if(courant->valeur==v){
```

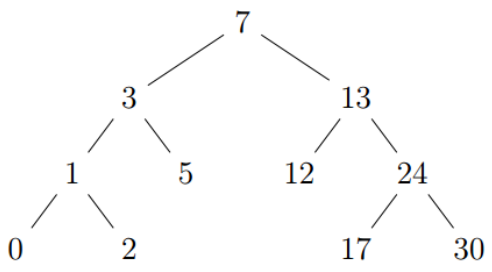
```
                    if(courant==precourant){ // courant est la tete de la liste
                    l->tete=courant->suivant; // nous devons changer la tête de la liste
                    free(courant); // libérer la mémoire
                    courant=l->tete;
                    precourant=courant;
                    }
                    else{
                            precourant->suivant=courant->suivant;
                            free(courant);
                            courant=precourant->suivant;
                    }
            else{
                    precourant=courant;
                    courant=courant->suivant;
            }
            }
    }

}
```

**Exercice 3 :**

// Question 1

- Un arbre binaire est un arbre où chaque nœud est connecté à deux sous-arbres (un sous arbre binaire gauche et un sous-arbre binaire droit).
- Un arbre binaire de recherche est un arbre binaire ordonné tel que pour tout nœud i : Toutes les valeurs du sous arbre gauche de i sont strictement inférieures à la valeur de i, et toutes les valeurs du sous arbre droit de i sont supérieures ou égales à la valeur de i.

// Question 2




// Question 3

- infixe, postfixe et préfixe, Le parcours infixe.

// Question 4

- infixe : 0 1 2 3 5 7 12 13 17 24 30
- postfixe : 0 2 1 5 3 12 17 30 24 13 7

- préfixe : 7 3 1 0 2 5 13 12 24 17 30

// Question 5
La structure de données d'un arbre binaire de recherche :

```
typedef struct nœud{
  int valeur;
  struct nœud *gauche;
  struct nœud *droit;
  } arbre;
```

---

```
void parcours_Infixe(arbre *racine){
   if(racine!=NULL){
   parcours_Infixe(racine->gauche);
   printf("%d \t", racine->donnee);
   parcours_Infixe(racine->droit);}}
```

---

```
void parcours_postfixe(arbre *racine){
  if(racine != NULL){
  parcours_postfixe(racine->gauche);
  parcours_postfixe(racine->droit);
  printf("%d \t", racine->donnee);}}
```

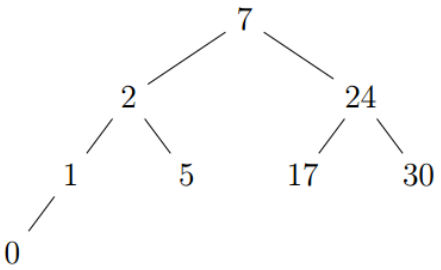
---

```
void parcours_prefixe(arbre *racine){
  if(racine!=NULL){
  printf("%d \", racine–>donnee);
  parcours_prefixe(racine–>gauche);
  parcours_prefixe(racine–>droit);}}
```

// Question 6
Pour supprimer le nœud *i* d'un arbre binaire de recherche, il faudra le rechercher. Une fois le nœud *i* trouvé, on se trouve dans une des situations suivantes :
**Cas 1** : Suppression d'une feuille. Libérer le nœud i.
**Cas 2** : Suppression d'un nœud avec un fils. Chaîner le père au fils de i (fils gauche de i ou fils droit de i) ensuite libérer le nœud i.
**Cas 3** : Suppression d'un nœud avec deux fils. Rechercher le plus proche prédécesseur ou successeur de i, soit p. Remplacer la partie donnée de i par la partie donnée de p. Supprimer le nœud p.

// Question 7

Le nœud 3 possède deux fils, donc il faut le remplacer avec son prédécesseur direct (le nœud le plus à droite de son fils gauche). Ensuite supprimer son prédécesseur direct en utilisant l'une des deux suppressions.


7
2
24
1
5
17
30
0