Exercice 1

Problème
Écrire l'algorithme d'un sous-programme qui permet de calculer la somme de deux polynômes, avec un polynôme représenté comme un tableau de N monômes triés par ordre croissant sur les degré de monômes.

En entrée :
$P_1(X) = 1 + 3x^{34} + 5\ x^{645}$
$P_2(X) = 3 + 4x^2 + 5\ x^{645} + 6\ x^{729}$
En sortie :
$P_3(X) = 4 + 4x^2 + 3x^{34} + 6\ x^{729}$

Monôme = Degré (Entier) x Coefficient (Réel)

Polynôme = Tableau de K monôme triés par ordre croissant sur les degré de monômes.

Résultat :
Polynôme de au plus n+m monômes triés par ordre croissant sur les degré de monômes.
Données :
P1 Polynôme de m monômes, triés par ordre croissant sur les degré de monômes.
P2: Polynôme de n monômes, triés par ordre croissant sur les degré de monômes.

Fonction sum(P1: Polynôme de m monômes, P2: Polynôme de n monômes) : Polynôme de n+m monômes
Début

```
i ← 1 //iterates over P1
j ← 1 //iterates over P2
h ← 1 //iterates over P
P ← créerPolynome()

TantQue ( i <= m ) ET ( j <= n) faire
        Si ( degré(P1[i ]) == degré (T2[j]) )alors
                C ← coefficient( P1[i] ) + coefficient( P2[i ] ) //holds the coefficient of the result .
                //NOTE that Si C ==0 should not be add
                Si ( C != 0 ) alors
                        P[h] ← créerMonome()
                         degré(P[h]) ← degré(P1[i])
                         coefficient(P[h]) ← C
                         h ← h+1; // we increment only Si we add the record to the array.
                finSi
                i ← i+1
                j ← j+1
        sinon
                Si ( degré(P1[i ]) < degré (T2[j]) ) alors
                        P[h] ← créerMonome()
                         degré(P[h]) ← degré(P1[i])
                         coefficient(P[h]) ← coefficient(P1[h])
                         i ← i+1
                         h ← h+1
                sinon
                        P[h] ← créerMonome()
                         degré(P[h]) ← degré(P2[i])
                         coefficient(P[h]) ← coefficient(P2[h])
                         j ← j+1
                         h ← h+1
                finSi
        finSi
Fait

TantQue ( j <= n) do
        P[h] ← créerMonome()
         degré(P[h]) ← degré(P2[j])
         coefficient(P[h]) ← coefficient(P2[j])
         h ← h+1
         j ← j+1
Fait

TantQue ( i <= m) do
        P[h] ← créerMonome()
         degré(P[h]) ← degré(P1[i])
         coefficient(P[h]) ← coefficient(P1[i ])
         h ← h+1
         i ← i+1
Fait
```

Fin.

Exercice 2

Problème
Fonctions insérer/supprimer en tête d'une liste simplement chaînée mais en considérant deux représentations chaînées de liste d'entiers.

V1 – La liste en tant que pointeur sur le premier élément de la liste

```
typedef struct elem {
        int value ;
        struct elem* next ;
} Element ;
typedef Element* ListOfInteger ;

ListOfInteger insertHead(ListOfInteger  l , Int i ) {
         Element * newel ;
         newel = (Element *) malloc ( sizeof ( Element ) ) ;
         newel->value = i;
         newel->next = l ;
         return newel ;
 }

l = insertHead(l,3) ;


ListOfInteger removeHead(ListOfInteger  l) {
        Element* second = NULL ;
        if (l!= NULL) {
                second = l→next ;
                free(l) ;
        }
        return second ;
}
```

V2 – La liste en tant que structure à deux champs (un pointeur sur le premier élément, la taille de la liste)

```
typedef struct elem {
        int value ;
        struct elem* next ;
} Element ;

typedef struct {
        Element* head ;
        int size ;
} ListOfInteger ;

ListOfInteger insertHead(ListOfInteger l , Int i ) {
         Element newel = NULL ;
         newel = (Element *) malloc( sizeof( Element ) ) ;
         newel->value = i;
         newel->next = l.head;

        l.size++ ;
        l.head = newel ;

        return l;
 }

ListOfInteger removeHead(ListOfInteger  l) {
        if (l.head!= NULL) {
                Element* oldHead= l.head ;
                l.head=l.head→next ;
                free(oldHead) ;
                l.size-- ;
        }
        return l ;
}
```

Exercice 2

Problème

Fonctions insérer en queue d'une liste simplement chaînée mais en considérant deux représentations chaînées de liste d'entiers.

V1 – La liste en tant que pointeur sur le premier élément de la liste

```
typedef struct elem {
        int value ;
        struct elem* next ;
} Element ;
typedef Element* ListOfInteger ;

Fonction insérerQueue(l : Liste<Entier>, e : Entier) : Liste<Entier>
```

```
Début
        newel : Element ← créerElement()
        valeur(newel) ← e
        succ(newel) ← INDEFINI

        Si estVide(l) alors
                l ← newel (ou tete(l) ← newel)
        sinon
                p : Element ← tete(l)
                TantQue non estVide(succ(p)) faire
                        p ← succ(p)
                fait
                succ(p) ←newel
        FinSi
        insérerQueue ← l
Fin.
```

V2 – La liste en tant que structure à deux champs (un pointeur sur le premier élément, la taille de la liste)

```
typedef struct elem {
        int value ;
        struct elem* next ;
} Element ;
```

Element = Valeur(Entier) x Suivant(Element)

```
typedef struct {
        Element* head ;
        int size ;
} ListOfInteger ;
```

ListOfInteger = Element x Taille(Entier)

```
Fonction insérerQueue(l : Liste<Entier>, e : Entier) : Liste<Entier>
Début
        newel : Element ← creerElement()
        valeur(newel) ← e
        succ(newel) ← INDEFINI

        Si estVide(l) alors
                tete(l) ← newel
                taille(l) ← 1
        sinon
                p : Element ← tete(l)
                TantQue non estVide(succ(p)) faire
                        p ← succ(p)
                fait
                succ(p) ←newel
                taille(l) ← taille(l) + 1
        FinSi
        insérerQueue ← l
Fin
```