Exercice 1

Problème

Écrire l'algorithme des sous-programmes Min et Max pour d'obtenir l'élément d'une liste contenant respectivement la valeur minimale, et la valeur maximale des codes produit d'une liste de Produits, sachant que Produit = Code x Price x Quantity = Entier x Réel x Réel
On retourne NULL si la liste est vide.

```
typedef struct {
	int code;
	float price;
	float quantity;
} Product;

typedef struct elem {
	Product value;
	struct elem* next;
} PElement;
```

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

```
typedef PElement* ListOfProduct;
```

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

```
typedef struct {
	PElement* head ;
	int size ;
} ListOfProduct ;
```

Fonction min(l : Liste<Produit>) : Element<Produit>
Début
	p : Element<Produit> ← tete(l)

	Si estVide(l) alors
		min ← VIDE
	sinon
			minE : Element<Produit> ← tete(l)
			p ← succ(p) ;
			TantQue non estVide(p) faire
				si code(valeurTete(p)) < code(valeur(minE)) alors
					minE ← p
				finSi
				p ← succ(p)
			fait
			min ← minE
	finSi
Fin.

```
Fonction max(l : Liste<Produit>) : Element<Produit>
Début
        p : Element<Produit> ← tete(l)

        Si estVide(l) alors
                max ← VIDE
        sinon
                        maxE : Element<Produit> ← tete(l)
                        p ← succ(p) ;
                        TantQue non estVide(p) faire
                                si code(valeurTete(p)) > code(valeur(maxE)) alors
                                        max ← p
                                finSi
                                p ← succ(p)
                        fait
                        max ← maxE
        finSi
Fin.
```

Exercice 2

Problème

Écrire l'algorithme du sous-programme SelectionSortMinBased pour trier une liste de produits par ordre croissant sur les code produits, sachant que
Produit = Code x Price x Quantity = Entier x Réel x Réel

```
Fonction SelectionSortMinBased(l : Liste<Produit>) : Liste<Produit>
Début
        minE : Element<Produit> ← INDEFINI | VIDE

        p : Element<Produit> ← tete(l)
        TantQue non estVide(p) faire
                minE ← min(p) //fonctionne en V1 uniquement, V2 faut créer une nouvelle liste
        fait
Fin
```

Listes doublement chaînées
Exercice 1

Défintion d'une liste doublement chaînée d'entiers

Version 1 – Liste en tant pointeur sur la tête

```
typedef struct elem {
        int value ;
        struct elem* next ;
        struct elem* prev ;
} DListElement ;
typedef DListElement* DList ;
```

Version 2 – Liste en tant que structure

```
typedef struct elem {
        int value ;
        struct elem* next ;
        struct elem* prev ;
} DListElement ;

typedef struct {
        DListElement head ;
        DListElement tail ;
        int size ;
} Dlist ;
```

Problème
Insérer un nouvel entier en tête d'une liste doublement chaînée

Version 1 – Liste en tant pointeur sur la tête

```
Fonction insertHead(l : Liste<Entier>, e:Entier) : Liste<Entier>
Début
        newel : DListElement<Entier> ← créerElement()
        value(newel) ← e
        suivant(newel) ← tête(l) OU juste l
        précédent(newel) ← INDEFINI

        Si (non estVide(l)) alors
                precedent(l) ← newel OU precedent(tete(l)) ← newel
        FinSi

        insertHead ← newel
Fin.
```

Version 2 – Liste en tant que structure

```
Fonction insertHead(l : Liste<Entier>, e:Entier) : Liste<Entier>
Début
        newel : DListElement<Entier> ← créerElement()
        value(newel) ← e
        suivant(newel) ← tête(l)
        précédent(newel) ← INDEFINI

        Si (non estVide(l)) alors
                precedent(tête(l)) ← newel
        sinon
                queue(l) ← newel
        FinSi

        tête(l) ← newel
        taille(l) ← taille(l) + 1
        insertHead ← l
Fin.
```

Exercice 2

Problème

Insérer un nouvel entier en queue d'une liste doublement chaînée

Version 1 – Liste en tant pointeur sur la tête

```
Fonction insertQueue(l : Liste<Entier>,e:Entier) : Liste<Entier>
Début
        newel : DListElement<Entier> ← créerElement()
        value(newel) ← e
        suivant(newel) ← INDEFINI
        précédent(newel) ← INDEFINI

        Si estVide(l) alors
                l ← newel OU tête(l) ← newel
        sinon
                p : DListElement ← tête(l)
                TantQue non estVide(suivant(p)) faire
                        p ← suivant(p)
                fait
                suivant(p) → newel
                précédent(newel) ← p
        FinSi

        insertQueue ← l
Fin.
```

Version 2 – Liste en tant que structure

```
Fonction insertQueue(l : Liste<Entier>, e:Entier) : Liste<Entier>
Début
        newel : DListElement<Entier> ← créerElement()
        value(newel) ← e
        suivant(newel) ← INDEFINI
        précédent(newel) ← INDEFINI

        Si estVide(l) alors
                tête(l) ← newel
        sinon
                suivant(queue(l)) → newel
                précédent(newel) ← queue(l)
        FinSi

        queue(l) ← newel
        taille(l) ← taille(l) + 1
        insertQueue ← l
Fin.
```

Exercice 3

Problème
Supprimer la tête d'une liste doublement chaînée

Version 1 – Liste en tant pointeur sur la tête

```
Fonction removeHead(l : Liste<Entier>) : Liste<Entier>
Début
        p : DListElement<Entier> ← INDEFINI

        Si non estVide(l) alors
                p ← suivant(tete(l)) ou suivant(l)
                Si non estVide(p) alors
                        precedent(p) ← INDEFINI
                finSi
                free(l) OU free(tete(l))
        finSi
        removeHead ← p
Fin.
```

Version 2 – Liste en tant que structure

```
Fonction removeHead(l : Liste<Entier>) : Liste<Entier>
Début
        p : DListElement<Entier> ← INDEFINI

        Si non estVide(l) alors
                p ← suivant(tete(l))
                Si non estVide(p) alors
                        precedent(p) ← INDEFINI
                finSi
                free(tête(l))
                tête(l) ← p

                Si taille(l) = 1 alors
                        queue(l) ← INDEFINI
                finSi

                taille(l) ← taille(l) - 1
        finSi

        removeHead ← l
Fin.
```